Rapport de stage de L3

Modèles continus de supply chains

Application à la modélisation d'une architecture de calcul parallèle

Benjamin Dadoun, Laurent Feuilloley, Siyu Zhang

2 juillet 2012

Équipe encadrante :
Florian De Vuyst, Francesco Salvarani, Daniel Bouche, Pascal Jaisson

Résumé

Au cours de ce stage, nous avons d'abord étudié l'application d'un modèle de supply chains à la modélisation d'une architecture de calcul parallèle. Cette approche, reposant essentiellement sur le concept de files d'attente, est intéressante par sa simplicité, et fait l'objet de plusieurs de nos modélisations. Néanmoins, elle ne semble pas suffisante pour modéliser précisément des architectures de calcul haute performance telles que nous les connaissons aujourd'hui. Dans un deuxième temps nous avons donc étudié un autre type de simulation, en nous inspirant des travaux récents sur l'évaluation des performances des processeurs.

Table des matières

Introduction

Une *supply chain*, ou « chaîne logistique », désigne le système des *flux* de ressources (organisations, personnes, technologies, activités, etc.) impliquées dans la mise à disposition de produits, depuis leur fournisseur (*supplier*) jusqu'au client final (*customer*). Évidemment, la gestion des supply chains constitue un enjeu prioritaire pour la productivité d'une entreprise. Dans un autre ordre d'idée, leur structure en « réseau de flux » rappelle celle d'une architecture d'ordinateurs massivement parallèles de type cluster, dans laquelle des données sont distribuées entre des milliers de processeurs interconnectés.

Dans ce document, nous présentons d'abord plusieurs modèles mathématiques liés à une modélisation des supply chains [ADR06, DGHP10]. Celle que nous décrivons, au chapitre 2, est principalement fondée sur la gestion de *files d'attente*. Au chapitre 3, nous étudions ensuite leur application à la modélisation d'une architecture de calcul parallèle. Ceci nous amènera aussi à proposer des modèles en rapport avec le calcul haute performance [WWP09, Wil08, DeV12], qui sont un peu plus éloignés des supply chains (chapitre 6).

1. Préliminaires mathématiques

Dans ce chapitre, sont introduits plusieurs outils mathématiques utilisés dans la modélisation des supply chains et rencontrés au cours de notre étude. Pour plus de détails, on pourra se référer à [DGHP10].

1.1. Lois de conservation

Le point de départ de la modélisation est d'assimiler l'évolution d'une supply chain à l'écoulement d'un *fluide* dans un réseau (graphe). Ceci permet d'introduire des modèles continus vérifiant une loi de conservation hyperbolique, comme c'est généralement le cas en mécanique des fluides.

Nous nous intéressons donc principalement à la conservation d'une « densité de matière » à une coordonnée d'espace $\rho(t,x) \in \mathbf{R}^d$, transportée par un flux $f(\rho) \in \mathbf{R}^d$. La variation $(\rho(t+\mathrm{d}t,x)-\rho(t,x))\mathrm{d}x$ de cette matière sur une portion d'abscisse infinitésimale $[x, x+\mathrm{d}x]$ entre les instants t et $t+\mathrm{d}t$ s'exprime par :

$$(\rho(t+\mathrm{d}t,x)-\rho(t,x))\mathrm{d}x = \underbrace{f(\rho(t,x))\mathrm{d}t}_{\text{ce qui rentre}} - \underbrace{f(\rho(t,x+\mathrm{d}x))\mathrm{d}t}_{\text{ce qui sort}}.$$

D'où l'équation de conservation de ρ :

$$\partial_t \rho + \partial_x f(\rho) = 0. \tag{1.1}$$

Dans le cas où $f : \mathbf{R}^d \to \mathbf{R}^d$ est régulière, l'équation (1.1) se réécrit :

$$\partial_t \rho + f'(\rho)\partial_x \rho = 0, \tag{1.2}$$

où $f'(\rho)$ est la différentielle de f en ρ. Lorsque les valeurs propres de $f'(\rho)$ sont réelles pour tout ρ, on dit que le système (1.2) est hyperbolique.

1.2. Schémas numériques

Pour calculer une solution approchée d'une équation aux dérivées partielles (EDP), on introduit un maillage en espace, de pas h, et en temps, de pas Δt (avec $h \to 0$ lorsque $\Delta t \to 0$). La solution approchée est alors calculée numériquement à partir d'une discrétisation par différences finies.

Une condition souvent nécessaire à la convergence de la solution numérique vers la solution exacte de l'équation continue est la **stabilité** du schéma. Cette propriété assure que la solution numérique n'« explose » pas lorsque les pas de discrétisation tendent vers 0.

1.2.1. Schéma explicite d'ordre 1

Le schéma de discrétisation le plus simple est sans doute le schéma explicite centré. Pour l'équation (1.1), le schéma est :

$$\frac{\rho_j^{n+1} - \rho_j^n}{\Delta t} - \frac{f(\rho_{j+1}^n) - f(\rho_{j-1}^n)}{2h} = 0. \tag{1.3}$$

Nous allons voir que le schéma explicite centré (1.3) n'est pas stable (au sens L^2) dans le cas où f est scalaire, c'est-à-dire dans le cas de l'équation de transport linéaire :

$$\partial_t u + a\partial_x u = 0, \quad a > 0. \tag{1.4}$$

Le schéma se réécrit donc :

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_{j+1}^n - u_{j-1}^n}{2h} = 0. \tag{1.5}$$

Proposition 1.2.1. *Le schéma explicite centré (1.5) n'est pas L^2-stable.*

Démonstration. Nous décrivons ici la méthode de von Neumann. L'équation semi-discrétisée de (1.5) est :

$$u^{n+1}(x) = u^n(x) - a\Delta t\frac{u^n(x+h) - u^n(x-h)}{2h} = 0,$$
$$u^n(x) = u_j^n, \quad x \in \left[jh - \frac{h}{2}, jh + \frac{h}{2}\right[.$$

L'idée de la méthode est d'exploiter l'égalité des normes $\|u^n\|_2 = \|\hat{u}^n\|_2$, où $\hat{u}^n$ est la transformée de Fourier de u^n :

$$\begin{aligned}\hat{u}^{n+1}(\xi) &\doteq \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} u^{n+1}(x)\exp(-\mathrm{i}\xi x)\mathrm{d}x \\ &= \hat{u}^n(\xi) - a\frac{\Delta t}{2h}(2\mathrm{i}\sin(h\xi))\hat{u}^n(\xi) \\ &= \mathcal{J}_{\Delta t,h}(\xi)\hat{u}^n(\xi),\end{aligned}$$

avec $\mathcal{J}_{\Delta t,h}(\xi) \doteq 1 - a\frac{\Delta t}{h}\mathrm{i}\sin(h\xi)$. Ainsi, quel que soit $\Delta t > 0$, nous avons

$$|\mathcal{J}_{\Delta t,h}(\xi)| = \sqrt{1 + a^2\frac{\Delta t^2}{h^2}} > 1$$

pour $\sin(h\xi) \neq 0$, donc le schéma (1.5) n'est pas L^2-stable ($\|u^n\|_2 \to +\infty$ quand $n \to +\infty$). □

On utilise de préférence un schéma décentré amont (*upwind*) :

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_j^n - u_{j-1}^n}{h} = 0. \tag{1.6}$$

Proposition 1.2.2. *Le schéma décentré amont (1.6) est L^2-stable lorsque $a\frac{\Delta t}{h} < 1$.*

Démonstration. Ici, $\hat{u}^{n+1}(\xi) = \left(1 - a\frac{\Delta t}{h}\left(1 - \exp(\mathrm{i}h\xi)\right)\right)\hat{u}^n(\xi)$, donc le schéma est L^2-stable lorsque que $Co \doteq a\frac{\Delta t}{h}$ (appelé nombre de Courant) est strictement inférieur à 1 (condition CFL[1]). □

1.2.2. Schéma explicite d'ordre 2

Considérons l'équation de la chaleur à une dimension :

$$\frac{\partial u}{\partial t} - \kappa\frac{\partial^2 u}{\partial x^2} = 0. \tag{1.7}$$

Le schéma explicite est :

$$u_j^{n+1} = u_j^n + \kappa\frac{\Delta t}{h^2}(u_{j+1}^n - 2u_j^n + u_{j-1}^n). \tag{1.8}$$

Proposition 1.2.3. *Si $\kappa\frac{\Delta t}{h^2} \leqslant \frac{1}{2}$, le schéma (1.8) est L^∞-stable.*

Démonstration. Notons $U^n = \left(u_j^n\right)_j$. Supposons $\beta \doteq \kappa\frac{\Delta t}{h} \leqslant \frac{1}{2}$. Nous avons $U^{n+1} = MU^n$, où

$$M = \begin{pmatrix} 1-2\beta & \beta & 0 & \dots & 0 \\ \beta & 1-2\beta & \beta & \ddots & \vdots \\ 0 & \beta & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 1-2\beta & \beta \\ 0 & \dots & 0 & \beta & 1-2\beta \end{pmatrix}$$

est une matrice positive vérifiant $\|M\|_\infty = 1$. Donc $U^n = M^n U_0$, et

$$\begin{aligned} \|U^n\|_\infty &\leqslant \|M^n\|_\infty \left\|U^0\right\|_\infty \\ &\leqslant \|M\|_\infty^n \left\|U^0\right\|_\infty \\ &= \left\|U^0\right\|_\infty. \end{aligned}$$

□

1.3. Solutions faibles

Il arrive que le phénomène étudié comporte des singularités, qui doivent donc naturellement être autorisées par le modèle. Un exemple de singularité est le congestionnement d'une file d'attente, que nous détaillons au chapitre 2.

Pour tenir compte de ces singularités, nous ne pouvons plus considérer la solution $\rho(t,x)$ comme une fonction au sens classique. Nous passons à une *formulation faible* : la

1. Courant-Friedrichs-Lewy.

solution recherchée est alors une distribution. Ainsi, nous disons que $\rho(t,x)$ est *solution faible* de (1.1) si, pour toute fonction test φ (typiquement, les fonctions $\mathcal{C}^\infty$ à support compact),

$$\int_0^{+\infty}\int_{-\infty}^{\infty} \varphi(t,x)\left\{\partial_t\rho(t,x)+\partial_x f(\rho(t,x))\right\}\mathrm{d}x\mathrm{d}t=0. \tag{1.9}$$

Bien sûr, les solutons « classiques » de (1.1) vérifient (1.9) (d'où l'appellation « faible »).

1.4. Méthode des caractéristiques

Une méthode de résolution des EDP est la *méthode des caractéristiques*, que nous illustrons ici dans le cas de l'équation de Burgers non visqueuse

$$\partial_t u + u\partial_x u = 0, \tag{1.10}$$

avec la donnée initiale

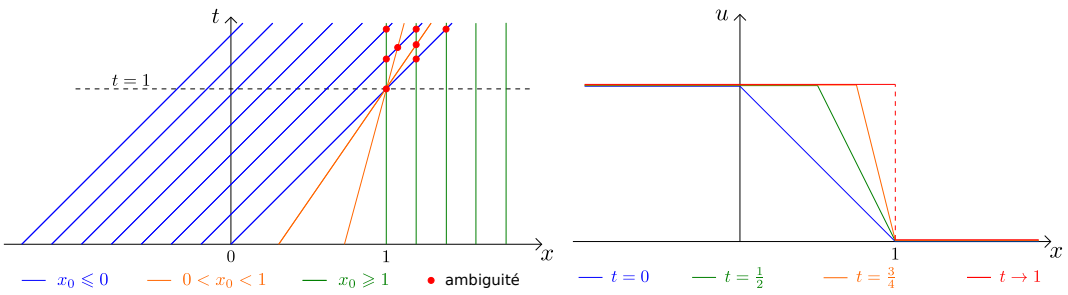
$$u_0(x)=\begin{cases}1 & \text{si } x\leqslant 0\\ 1-x & \text{si } 0<x<1\\ 0 & \text{si } x\geqslant 0\end{cases}. \tag{1.11}$$

Les courbes caractéristiques sont les courbes $(t(s),x(s))$ sur lesquelles u est constante. En posant $g(s)=u(t(s),x(s))$, on a

$$g'(s)=0 \iff t'(s)\partial_t u(t(s),x(s))+x'(s)\partial_x u(t(s),x(s))=0$$

ce qui est vérifié d'après (1.10) lorsque $t'(s)=1$ et $x'(s)=g(s)=g(0)$. Alors $t(s)=s$ et $x(s)=u_0(x_0)s+x_0$. Ainsi, les courbes caractéristiques sont des droites (voir figure 1.4.1) :

- pour $x_0\leqslant 0$, u est constante et vaut 1 sur la droite $t=x-x_0$;
- pour $0<x_0<1$, u est constante et vaut $1-x_0$ sur la droite $t=\frac{1}{1-x_0}(x-x_0)$;
- pour $x_0\geqslant 1$, u est constante et vaut 0 sur la droite $x=x_0$.



(a) Droites caractéristiques (b) Solution pour $t<1$

FIGURE 1.4.1.: Méthode des caractéristiques

Graphiquement, on observe que la solution est déterminée sans problème pour tout $t<1$, mais que les caractéristiques se coupent au-delà de la ligne $t=1$: il y a perte de régularité en temps fini.

2. Modèles simples de files d'attente

Nous présentons un modèle continu de supply chains permettant leur description dans le temps, initialement introduit par [ADR06]. Ce modèle formule, à partir d'une simulation par événements discrets, une loi de conservation $\partial_t \rho + \partial_x \min\{V\rho, \mu\} = 0$ pour la densité d'objets, la fonction de flux f étant limitée par les capacités maximales des fournisseurs.

2.1. Motivation

Une approche efficace pour modéliser les supply chains est l'approche dite *simulation par événements discrets*. L'idée principale est de calculer les temps de transport d'objets d'un point de départ (fournisseurs) vers leur destination (clients). Ce temps dépend de la demande des consommateurs et des politiques des fournisseurs. Dans la suite, on considère le cas d'une supply chain linéaire constituée de fournisseurs $S_1, \ldots, S_M$ ayant chacun un temps de traitement T et une capacité μ.

Les modèles de simulation par événements discrets décrivent de façon très précise les réseaux d'approvisionnement, mais les calculs qu'ils demandent deviennent extrêmement importants dans le cas de nombreux fournisseurs et objets. Ceci étant, Armbruster, Degond et Ringhofer [ADR06] ont pu en déduire un modèle continu qui n'a pas ce défaut.

2.2. Simulation par événements discrets

Considérons une supply chain linéaire constituée de M « processeurs », le processeur m étant relié au processeur $m+1$. Chaque processeur m est caracterisé par un temps de traitement $T(m)$ et une capacité $\mu(n, m)$ (dépendant éventuellement de l'objet n à traiter), et dispose d'une file d'attente FIFO (premier arrivé, premier sorti) non bornée dans laquelle les objets sont placés avant d'être traités. Le choix le plus simple est de prendre $\mu(n, m) = \frac{1}{T(m)}$, *i.e.* le processeur traite constamment un seul objet par unité de temps.

On note a_n^m le temps d'arrivée de l'objet n au début du processeur m, b_n^m le temps d'entrée dans le processeur et e_n^m le temps de sortie du processeur.

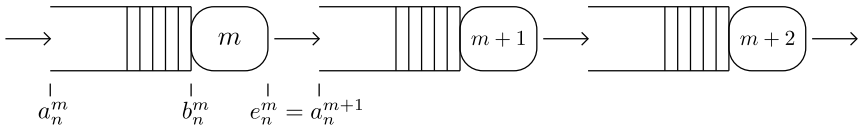


FIGURE 2.2.1.: Files d'attente et processeurs reliés en série

Si la file d'attente est vide, l'objet n entre immédiatement dans le processeur m et dans ce cas le temps d'entrée correspond au temps d'arrivée : $b_n^m = a_n^m$. En revanche, si la file d'attente n'est pas vide, l'objet n doit attendre avant d'entrer dans le processeur. On a donc les relations :

$$b_n^m = \max\left\{a_n^m, b_{n-1}^m + \frac{1}{\mu(n-1,m)}\right\}, \tag{2.1}$$

$$e_n^m = b_n^m + T(m). \tag{2.2}$$

Et on en déduit :

$$e_n^m = \max\left\{a_n^m + T(m), e_{n-1}^m + \frac{1}{\mu(n-1,m)}\right\}, \quad m = 0, 1, \ldots, \quad n \geqslant 1. \tag{2.3}$$

Dans le but d'obtenir une relation de récurrence sur les temps d'arrivée, on renote a_n^m en $\tau(n,m)$ (donc $e_n^m = \tau(n, m+1)$). La relation (2.3) se transforme en

$$\tau(n,m+1) = \max\left\{\tau(n,m) + T(m), \tau(n-1,m+1) + \frac{1}{\mu(n-1,m)}\right\}, \quad m = 0, 1, \ldots, \quad n \geqslant 1. \tag{2.4}$$

Pour bien définir les relations de récurrence, on ajoute les conditions initiale et limite

$$\tau(0,m) = \tau^I(m), \quad m = 0, 1, \ldots, \tag{2.5a}$$

$$\tau(n,0) = \tau^A(n), \quad n \geqslant 0. \tag{2.5b}$$

On peut aussi supposer que les files d'attente sont initialement vides :

$$f^A(\tau^A(n)) = \frac{1}{\tau^A(n+1) - \tau^A(n)}, \tag{2.6a}$$

$$\tau^I(m+1) = \tau^I(m) + T(m), \tag{2.6b}$$

où le flux d'entrée f^A est le taux d'objets arrivant au processeur $m = 0$.

Étant donné $\tau(n,m)$, la conservation du nombre d'objets peut être exprimée grâce au concept de courbes de Newell (N-courbes). En utilisant la fonction de Heaviside

$$H(y) = \begin{cases} 0 & \text{si } y < 0 \\ 1 & \text{si } y \geqslant 0 \end{cases},$$

la N-courbe

$$U(t,m) = \sum_{n=0}^{\infty} H(t - \tau(n,m)), \quad t > 0, \quad m = 0, 1, \ldots, \tag{2.7}$$

désigne le nombre d'objets arrivés au processeur m jusqu'au temps t. Le flux au processeur m est alors

$$F(t,m) \doteq \frac{\mathrm{d}}{\mathrm{d}t} U(t,m) = \sum_{n=0}^{\infty} \delta(t - \tau(n,m)), \tag{2.8}$$

et la quantité

$$W(t,m) \doteq U(t,m) - U(t,m+1), \quad m = 0, 1, \ldots, \tag{2.9}$$

correspond au nombre d'objets au niveau du processeur m au temps t. Ainsi, grâce à (2.8) et (2.9),

$$\frac{\mathrm{d}}{\mathrm{d}t} W(t,m) = F(t,m) - F(t,m+1). \tag{2.10}$$

Remarquons, pour conclure cette partie de modélisation discrète, que le temps de calcul d'une simulation est au moins linéaire en le nombre de processeurs et d'objets à traiter, ce qui peut-être problématique et encourage une approche continue.

2.3. Loi de conservation restreinte par les capacités

L'objectif de cette section est de remplacer asymptotiquement la relation (2.4) par une loi de conservation simple liant la densité de produits ρ et le flux f. Il s'agit dans un premier temps d'obtenir des expressions approximatives de ces quantités.

2.3.1. Densité et flux approximatifs

Plaçons chaque processeur m à une abscisse virtuelle x_m, vue comme le « niveau de traitement » d'un objet dans tout le système : $0 = x_0 < x_1 < \ldots < x_M = X$, et notons désormais $F(t, x_m)$ au lieu de $F(t,m)$. D'après (2.8), on a, pour toute fonction test ψ,

$$\begin{aligned} \int_{\tau^I(m)}^{\infty} \psi(t) F(t, x_m) \mathrm{d}t &= \sum_{n=0}^{\infty} \int_{\tau^I(m)}^{\infty} \psi(t) \delta(t - \tau(n,m)) \mathrm{d}t \\ &= \sum_{n=0}^{\infty} \psi(\tau(n,m)). \end{aligned}$$

En réécrivant ceci comme une somme de Riemann,

$$\int_{\tau^I(m)}^{\infty} \psi(t) F(t, x_m) \mathrm{d}t = \sum_{n=0}^{\infty} \psi(\tau(n,m)) \Delta_n \tau(n,m) f(\tau(n,m), x_m), \tag{2.11}$$

le flux $f(t,x)$ pour $t = \tau(n,m)$ et $x = x_m$ étant déterminé comme l'inverse de la différence de temps $\Delta_n \tau(n,m) \doteq \tau(n+1,m) - \tau(n,m)$:

$$f(\tau(n,m), x_m) = \frac{1}{\Delta_n \tau(n,m)}, \quad n \geqslant 0. \tag{2.12}$$

Sur une échelle de temps où $\tau(n,m)$ est petit, l'équation (2.11) devient

$$\int_{\tau^I(m)}^{\infty} \psi(t)F(t,x_m)\mathrm{d}t \approx \int_{\tau^I(m)}^{\infty} \psi(t)f(t,x_m)\mathrm{d}t.$$

Donc il est légitime de considérer (2.15) comme la définition du flux approximatif, donné à chaque niveau de traitement $x = x_m$.

Pour trouver maintenant une expression approximative de la densité ρ de produits par niveau de traitement, on considère le cas où les temps d'arrivée seraient distribués de façon continue, comme s'ils étaient donnés par une fonction $\tau(y,x)$. La relation (2.15) se réécrit :

$$f(\tau(y,x),x) = \frac{1}{\partial_y \tau(y,x)}. \tag{2.13}$$

Or, par conservation de la masse,

$$\frac{\mathrm{d}}{\mathrm{d}x} f(\tau(y,x),x) = 0,$$

c'est-à-dire

$$\frac{\partial}{\partial t}\underbrace{\left(f(\tau(y,x),x)\frac{\partial}{\partial x}\tau(y,x)\right)}+\frac{\partial}{\partial x}f(\tau(y,x),x) = 0.$$

Ceci nous incite à poser $\rho(y,x) = f(\tau(y,x),x)\frac{\partial}{\partial x}\tau(y,x)$ pour satisfaire une loi de conservation de la forme $\partial_t\rho + \partial_x f = 0$. Grâce à (2.13), on peut alors dire que

$$\rho = \frac{\partial_x \tau}{\partial_y \tau}. \tag{2.14}$$

D'un point de vue discret, la densité approximative de produits est donc

$$\rho(\tau(n,m+1),x_m) = \frac{\tau(n+1,m+1)-\tau(n+1,m)}{h_m(\tau(n+1,m+1)-\tau(n,m+1))}, \tag{2.15}$$

où $h_m \doteq x_{m+1} - x_m$ est le pas en espace.

Nous allons voir que les flux et densité approximatifs (2.12) et (2.15) ainsi définis satisfont une version discrétisée de la loi de conservation.

Théorème 2.3.1. *Si les relations (2.4), (2.12) et (2.15) sont satisfaites, alors le flux approximatif peut être écrit en fonction de la densité approximative :*

$$f(\tau(n,m),x_m) = \min\left\{\mu(n,m-1), \frac{h_{m-1}\rho(\tau(n,m),x_{m-1})}{T(m-1)}\right\}, \quad m = 1,\dots,M, \quad n \geqslant 0. \tag{2.16}$$

Démonstration. Nous proposons ici une preuve plus simple que celle donnée dans [ADR06]. Avec

$$\Delta_n\tau(n,m) \doteq \tau(n+1,m) - \tau(n,m),$$
$$\Delta_m\tau(n,m) \doteq \tau(n,m+1) - \tau(n,m),$$

nous pouvons réécrire la relation (2.4),

$$\tau(n+1,m) = \max\left\{\tau(n+1,m-1) + T(m-1), \tau(n,m) + \frac{1}{\mu(n,m-1)}\right\},$$

en

$$0 = \min\left\{\Delta_m\tau(n+1,m-1) - T(m-1), \Delta_n\tau(n,m) - \frac{1}{\mu(n,m-1)}\right\},$$

ce qui permet d'affirmer que $\Delta_m\tau(n+1,m-1) \geqslant T(m-1)$, $\Delta_n\tau(n,m) \geqslant \frac{1}{\mu(n,m-1)}$, et qu'il y a égalité dans au moins une de ces deux inégalités.
Avec

$$g(n,m) \doteq \min\left\{\mu(n,m-1), \frac{h_{m-1}\rho(\tau(n,m), x_{m-1})}{T(m-1)}\right\},$$

nous avons donc, compte tenu des hypothèses faites,

$$\begin{aligned} g(n,m) &= \min\left\{\mu(n,m-1), \frac{1}{T(m-1)}\frac{\Delta_m\tau(n+1,m-1)}{\Delta_n\tau(n,m)}\right\} \\ &= \frac{1}{\Delta_n\tau(n,m)}\min\left\{\mu(n,m-1)\Delta_n\tau(n,m), \frac{\Delta_m\tau(n+1,m-1)}{T(m-1)}\right\} \\ &= \frac{1}{\Delta_n\tau(n,m)} \\ &= f(\tau(n,m), x_m), \end{aligned}$$

et la relation (2.16) est alors démontrée. □

L'avantage de la relation (2.16) par rapport à (2.8) est qu'elle ne fait pas intervenir les temps d'arrivée de tous les produits. Dans la suite, on cherche à savoir si, et dans quelle mesure, cette loi de conservation discrète peut être assimilée asymptotiquement à une loi de conservation de la forme $\partial_t\rho + \partial_x f = 0$.

2.3.2. Validité asymptotique

On se place désormais dans le cas où le nombre de processeurs est très grand, *i.e.* $M \to \infty$. Nous verrons comment se passer de cette hypothèse légèrement irréaliste en section §2.3.3.

Adimensionnement. Définissant le temps moyen de traitement

$$T_0 \doteq \frac{1}{M}\sum_{m=0}^{M-1} T(m),$$

MT_0 correspond au temps que met un objet pour parcourir tout le système sans jamais attendre (files d'attente vides). On introduit alors les variables adimensionnées suivantes :

$$\tau(n,m) = MT_0\tau_s(n,m), \quad T(m) = T_0T_s(x_m), \quad \mu(n,m) = \frac{\mu_s(\tau_s(n,m+1),x_m)}{T_0}. \tag{2.17}$$

En reprenant les équations (2.4) et (2.5a), (2.5b) que satisfait $\tau(n,m)$, on déduit (avec $\varepsilon = \frac{1}{M}$) :

$$\begin{gathered}\tau_s(n+1,m+1) = \max\left\{\tau_s(n+1,m) + \varepsilon T_s(x_m), \tau_s(n,m+1) + \frac{\varepsilon}{\mu_s(\tau_s(n,m+1),x_m)}\right\},\\ \tau_s(n,0) = \tau_s^A(n), \quad n \geqslant 0, \quad \tau_s(0,m) = \tau_s^I(m), \quad m = 0,1,\ldots,\end{gathered} \tag{2.18}$$

où les conditions initiale et limite $\tau_s^I(m)$ et $\tau_s^A(n)$ sont adimensionnées de la même façon que $\tau(n,m)$. Par ailleurs, il est raisonnable de supposer que les $\Delta\tau$ sont de l'ordre de T_0, on pose donc

$$\begin{aligned}\Delta_n\tau(n,m) &= T_0\Delta_{ns}\tau_s(n,m),\\ \Delta_m\tau(n,m) &= T_0\Delta_{ms}\tau_s(n,m),\end{aligned}$$

de sorte que

$$\begin{aligned}\tau_s(n+1,m) &= \tau_s(n,m) + \varepsilon\Delta_{ns}\tau_s(n,m),\\ \tau_s(n,m+1) &= \tau_s(n,m) + \varepsilon\Delta_{ms}\tau_s(n,m).\end{aligned}$$

Il reste à donner les expressions adimensionnées du flux f et de la densité ρ. On choisit, conformément à (2.12) et (2.15),

$$\begin{aligned}f(t,x) &= \frac{1}{T_0}f_s\left(\frac{t}{MT_0},x\right),\\ \rho(t,x) &= \frac{M}{X}\rho_s\left(\frac{t}{MT_0},x\right),\end{aligned}$$

ce qui donne :

$$f_s(\tau(n,m),x_m) = \frac{1}{\Delta_{ns}\tau_s(n,m)}, \tag{2.19}$$

$$\rho_s(\tau_s(n,m+1),x_m) = \frac{\varepsilon X\Delta_{ms}\tau_s(n+1,m)}{h_m\Delta_{ns}\tau_s(n,m+1)}. \tag{2.20}$$

Avec ces nouvelles notations, (2.16) se lit :

$$f_s(\tau_s(n,m),x_m) = \min\left\{\mu_s(\tau_s(n,m),x_{m-1}), \frac{h_{m-1}\rho_s(\tau_s(n,m),x_{m-1})}{\varepsilon X T_s(x_{m-1})}\right\}. \tag{2.21}$$

Par simplicité, on ramène à 1 la vitesse de propagation dans (2.21) en posant

$$h_m = \varepsilon X T_s(x_m) = \frac{XT(m)}{\sum_{m'=0}^{M-1} T(m')}, \quad m = 0, 1, \dots.$$

Par simplicité encore, nous omettons désormais l'indice s puisque nous ne travaillerons plus qu'avec des quantités adimensionnées.

Interpolation et formulation faible. La difficulté majeure dans le passage à la limite $M \to \infty$ (ou de façon équivalente $\varepsilon \to 0$) est que le flux f peut devenir discontinu. Supposons par exemple que le processeur en x_m travaille plus vite que le processeur en x_{m+1}. Alors une (longue) file d'attente risque de se former [1] au niveau de ce dernier processeur, et à la limite des masses de Dirac vont apparaître car la masse doit être conservée. C'est pourquoi la densité limite sera une distribution et non une fonction au sens classique.

En tenant compte des capacités maximales, on s'attend à obtenir l'équation de conservation

$$\partial_t \rho + \partial_x f = 0, \quad f(t,x) = \min\left\{\rho(t,x), \mu(t,x)\right\}, \quad f(t,0) = f^A(t), \tag{2.22}$$

avec la condition initiale $\rho(0,x) = \rho_0(x)$. On a vu que $W(t,m) \doteq U(t,m) - U(t,m+1)$ correspondait au nombre d'objets au niveau du processeur m, donc la densité en x sera $\rho(t,x) = -\partial_x u(t,x)$, où $u(t,x)$ est une approximation de la N-courbe $U(t,m)$. En intégrant l'équation précédente par rapport à x, on obtient la loi de conservation relative à $u(t,x)$:

$$\begin{aligned} &\partial_t u(t,x) - \min\left\{\mu(t,x), -\partial_x u(t,x)\right\} = 0, \\ &\lim_{x\to 0^+} u(t,x) = g^A(t), \quad \frac{\mathrm{d}}{\mathrm{d}t} g^A(t) = f^A(t). \end{aligned} \tag{2.23}$$

Les variables en jeu sont pour l'instant connues aux points de coordonnées $(\tau(n,m), x_m)$. On les prolonge à un domaine continu grâce à une interpolation par paliers :

$$f_1(t,x_m) = f(\tau(n,m),x_m), \qquad \tau(n,m) \leqslant t < \tau(n+1,m) \tag{2.24a}$$

$$\rho_1(t,x_m) = \rho(\tau(n-1,m+1),x_m), \qquad \tau(n-1,m+1) \leqslant t < \tau(n,m+1) \tag{2.24b}$$

$$f^A(t) = \frac{1}{\Delta_n \tau^A(n)}, \qquad \tau^A(n) \leqslant t < \tau^A(n+1) \tag{2.24c}$$

1. De tels « étranglements » (*bottlenecks*) peuvent même arriver au début de la chaîne si le flux d'entrée est supérieur à la capacité du premier processeur.

et

$$u_1(t, x_0) = \int_{\tau(0,0)}^{t} f^A(s)\mathrm{d}s, \tag{2.25a}$$

$$u_1(t, x_{m+1}) = u_1(t, x_m) - \frac{h_m}{X}\rho_1(t, x_m), \tag{2.25b}$$

pour les paliers en temps, puis

$$f_2(t, x) = f_1(t, x_{m+1}), \tag{2.26a}$$

$$\tau_2^I(x) = \tau^I(m+1), \quad x_m \leqslant x < x_{m+1} \tag{2.26b}$$

$$u_2(t, x) = u_1(t, x_{m+1}), \tag{2.26c}$$

pour les paliers en espace.

Le théorème suivant énonce que u_2, f_2 satisfont une version faible de (2.22).

Théorème 2.3.2. *Si les délais $\Delta_m\tau(n, m)$ et $T(x_m)$ sont bornés (dans ce cas $h_m = \mathcal{O}(\varepsilon)$), alors pour $\varepsilon \to 0$, les fonctions u_2, f_2 vérifient :*

$$\begin{aligned} \partial_t u_2(t, x) = f_2(t, x), 0 < x < X,\ t > \tau_2^I(x) \\ u_2(\tau_2^I(x), x) = 0, \\ \lim_{x \to 0^+} u_2(t, x) = \int_{\tau_2(0,0)}^{t} f^A(s)\mathrm{d}s. \end{aligned}$$

faiblement en x et t.

Le lecteur trouvera notre version de la démonstration de ce théorème en appendice A.

2.3.3. Processeurs virtuels

Le théorème 2.3.2 est intéressant puisqu'il valide un modèle continu simple de supply chains, mais une hypothèse peu réaliste pour l'appliquer est que le nombre de *suppliers* (processeurs) doit être infini ($M \to \infty$). [ADR06, DGHP10] ont néanmoins montré comment « contourner » ce problème : si l'on dispose d'un petit nombre de processeurs $1, \ldots, M$, de capacités respectives $\mu_1, \ldots, \mu_M$, l'idée est de démultiplier chaque processeur j en K sous-processeurs virtuels de même capacité $\frac{\mu_j}{K}$. Ce faisant, on augmente suffisamment le nombre de processeurs pour pouvoir appliquer le théorème, sans toutefois modifier la configuration de la chaîne.

2.3.4. Exemple de solution

La loi de conservation (2.22) admet une solution exacte dans le cas d'un unique étranglement. Supposons que la chaîne se compose de processeurs « rapides » (de capacité

constante μ_1), sur l'intervalle $\left[0, \frac{1}{2}\right]$, suivis de processeurs « lents » (de capacité $\mu_2 < \mu_1$), sur l'intervalle $\left[\frac{1}{2}, 1\right]$, et choisissons un flus d'arrivée $f^A(t)$ intermédiaire :

$$\mu(x) = \begin{cases} \mu_1 & \text{pour } 0 < x < \frac{1}{2} \\ \mu_2 & \text{pour } \frac{1}{2} < x < 1 \end{cases}, \quad \mu_2 < f^A(t) < \mu_1.$$

La solution $\rho(t,x)$ sera donc la somme d'une fonction « classique » $\rho_c(t,x)$, comportant une discontinuité à $x = \frac{1}{2}$, et d'une fonction de Dirac de la forme $q(t)\delta\left(x - \frac{1}{2}\right)$, décrivant le phénomène de congestion à l'abscisse $x = \frac{1}{2}$. De fait, ρ_c satisfait une équation de transport linéaire à vitesse constante (voir section §1.2.1) :

$$\partial_t \rho_c + \partial_x \rho_c = 0, \ x \in \left]0, \frac{1}{2}\right[\cup \left]\frac{1}{2}, 1\right[, \quad \rho_c(t,0) = f^A(t), \quad \rho_c\left(t, \frac{1}{2}^+\right) = \mu_2,$$

ce qui peut être résolu via la méthode des caractéristiques (voir section §1.4). On obtient :

$$\rho_c(t,x) = \begin{cases} f^A(t-x) & \text{pour } 0 < x < \frac{1}{2} \\ \mu_2 & \text{pour } \frac{1}{2} < x < 1 \end{cases}. \tag{2.28}$$

Pour trouver $q(t)$, nous exprimons que $\rho(t,x) = \rho_c(t,x) + q(t)\delta(x - \frac{1}{2})$ est solution faible de (2.22). Pour toute fonction test $\phi(x)$,

$$\int_0^1 \phi(x) \left(\partial_t \rho(t,x) - \partial_x \min\{\mu(x), \rho(t,x)\}\right) \mathrm{d}x = 0,$$

c'est-à-dire, par une intégration par parties,

$$\int_0^1 \left(\phi(x)\partial_t \rho(t,x) - \min\{\mu(x), \rho(t,x)\}\partial_x \phi(x)\right) \mathrm{d}x = \phi(0) f^A(t) - \phi(1) \min\left\{\mu_2, \rho(t,1)\right\}.$$

En intégrant de nouveau par parties séparément sur les intervalles $\left]0, \frac{1}{2}\right[$ et $\left]\frac{1}{2}, 1\right[$, nous obtenons :

$$\begin{aligned} \int_0^1 \phi(x)\partial_t \rho_c(t,x)\mathrm{d}x + q'(t)\phi\left(\frac{1}{2}\right) + \int_0^1 \phi(x)\partial_x \min\{\mu(x), \rho_c(t,x)\}\mathrm{d}x \\ - \min\left\{\mu_1, \rho_c\left(t, \frac{1}{2}^-\right)\right\}\phi\left(\frac{1}{2}\right) + \min\left\{\mu_1, \rho_c(t,0)\right\}\phi(0) + \min\left\{\mu_2, \rho_c\left(t, \frac{1}{2}^-\right)\right\}\phi\left(\frac{1}{2}\right) \\ = \phi(0) f^A(t), \end{aligned}$$

ce qui équivaut à, puisque $\rho_c(t,x) < \mu(x)$ et $\rho_c(t,0) = f^A(t)$,

$$q'(t) = f^A\left(t - \frac{1}{2}\right) - \mu_2. \tag{2.29}$$

Ainsi, loin de $x = \frac{1}{2}$, la solution est donnée par (2.28), et la force du congestionnement de la file d'attente est donnée par (2.29).

2.4. Simulation numérique

Les résultats précédents ont fait l'objet d'une simulation sous MATLAB. L'objectif est de valider le passage au continu décrit en section §2.3. Reprenant [ADR06], nous avons traité le cas de 3 processeurs définis comme suit :

$$\mu(t,x) = \begin{cases} 15 & 0 < x < 0,2 \\ 10 & 0,2 < x < 0,8 \\ 15 & 0,8 < x < 1 \end{cases}.$$

Choisissons aussi un flux d'entrée $f^A(t)$ autour de ces différentes valeurs : d'abord au-dessous de 10, puis entre 10 et 15, puis au-dessus de 15, et enfin au-dessous de 10.

Les résultats en figure 2.4.2 montrent que l'on observe bien un étranglement au niveau du deuxième processeur $x \in [0.2, 0.8]$, qui est plus lent que le premier, ainsi qu'un étranglement entre les instants $t = 60$ et $t = 80$ au niveau du premier processeur, lorsque le flux d'entrée est supérieur à $\mu_1 = 15$.

De plus, les valeurs obtenues par le modèle continu, calculées à partir d'une discrétisation de (2.22), correspondent sensiblement à celles obtenues par le modèle discret, calculées à partir des temps $\tau(n,m)$. Notons que le temps de calcul est aussi beaucoup plus court (moins d'une seconde pour le modèle continu, contre une quinzaine de secondes pour le modèle discret). Le modèle continu semble donc satisfaisant.

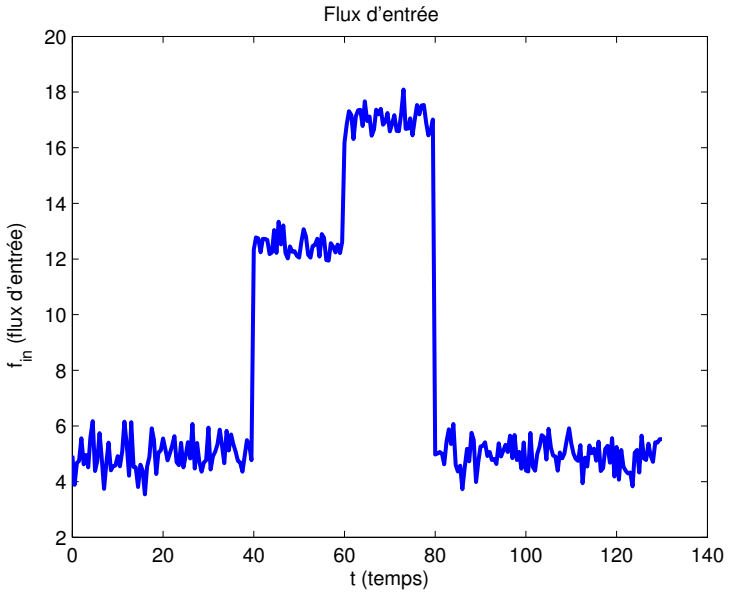


FIGURE 2.4.1.: Flux d'entrée

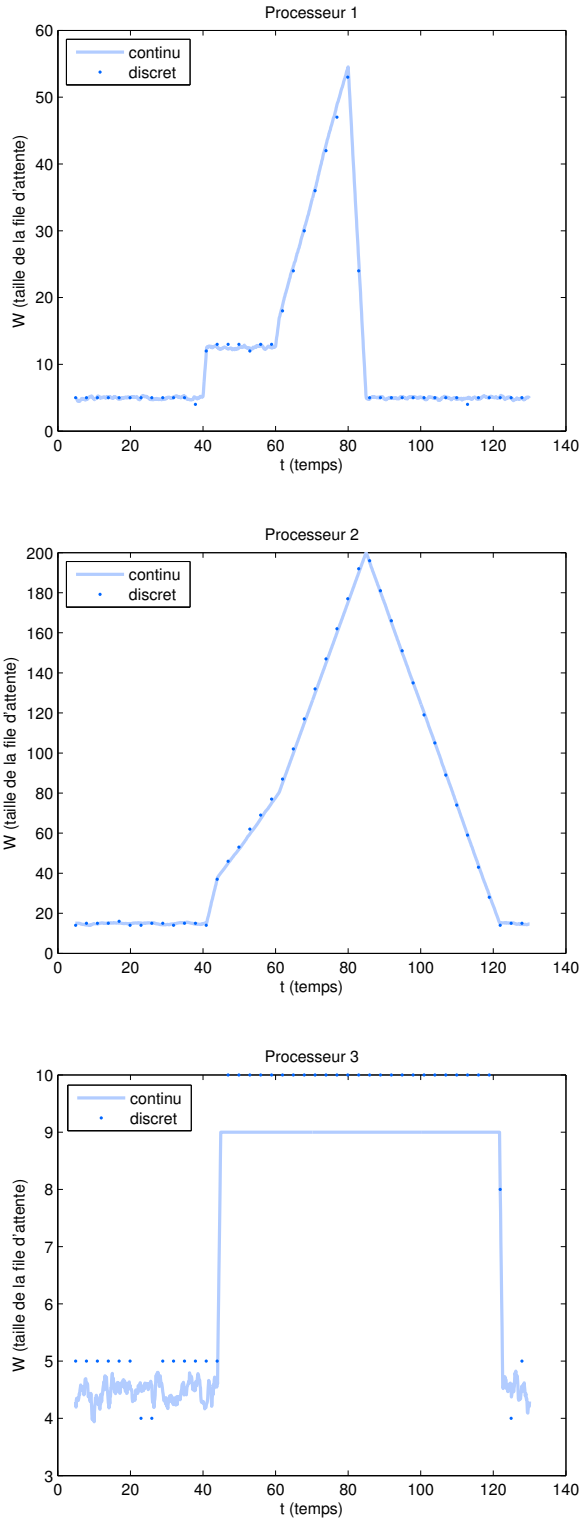


FIGURE 2.4.2.: État des files d'attente

3. Modélisation d'unités en parallèle (MUP)

Code associé : MUP.m.

3.1. Introduction

Dans cette partie, on présente un modèle utilisant des files d'attente en parallèle. L'idée est, en quelque sorte, d'utiliser les outils développés pour les supply chains mais dans un contexte de traitement parallèle et non séquentiel. À mesure de la construction du modèle, on est obligé de s'éloigner d'un modèle d'architecture parallèle ; ceci est discuté à la fin de cette partie.

3.2. Le modèle

On s'intéresse à des unités en parallèle, ayant chacune une file d'attente. La grandeur étudiée est la longueur de ces files d'attente. On suppose que des objets arrivent dans les files d'attente depuis une source extérieure. Les objets traités peuvent, ou bien ré-entrer dans le système en se mettant dans la file d'attente d'une des deux unités voisines, ou bien sortir définitivement.

3.3. Définition des grandeurs

On définit (voir figure 3.3.1) :
- $(l_k)_{1\leqslant k\leqslant n}$, les longueurs des files d'attente des unités (numérotées de 1 à n) ;
- f_k, le flux allant de l'usine k vers l'usine $k+1$ (positif ou négatif) ;
- e_k, le flux venant de l'extérieur qui rentre dans l'usine k ;
- s_k, le flux qui sort définitivement au niveau de l'usine k.

3.4. Les équations

3.4.1. Équations principales

On a le bilan suivant sur chaque file k :

$$\frac{\mathrm{d}l_k}{\mathrm{d}t} = e_k + f_{k-1} - f_k - s_k. \tag{3.1}$$

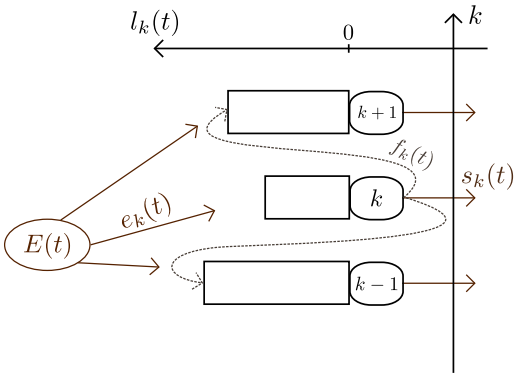


FIGURE 3.3.1.: Grandeurs en jeu

Pour des flux, entrée et sortie quelconques, le raisonnement s'arrête ici : on peut faire des modélisations discrètes et prévoir le comportement. On essaie d'utiliser les outils mathématiques décrits au chapitre 2 et en particulier, le passage au continu. On propose donc une série de choix qui peuvent se comprendre comme étant une politique (arbitraire) de gestion des unités.

On souhaite que le flux se déplace des usines les plus engorgées vers les usines les moins engorgées pour optimiser le traitement. On suggère donc l'expression suivante :

$$f_k = -\lambda(l_{k+1} - l_k). \tag{3.2}$$

Concernant le flux de sortie, on le choisit proportionnel à l_k :

$$s_k = hl_k. \tag{3.3}$$

3.4.2. Équations alternatives

On pourrait penser à d'autres politiques et fixer d'autres lois. En particulier si on veut un peu plus coller au problème de bande passante évoqué dans le modèle *Roofline*, on peut donner une borne maximum au flux inter-unités et à la sortie. On peut aussi penser à une fonction de sortie constante (l'équation utilisée précédemment dit que « plus il y a de travail à faire, plus le traitement est rapide », ce qui est discutable).

3.4.3. Analogie avec la thermodynamique

L'avantage des équations (3.1), (3.2), (3.3) est que l'analogie avec la thermodynamique est directe. Si on passe à une abscisse continue, on obtient l'équation (avec x l'abscisse de l'usine dont la version discrète est k)

$$\frac{\partial l(x,t)}{\partial t} = \lambda \frac{\partial^2 l(x,t)}{\partial x^2} + e(x,t) - hl(x,t). \tag{3.4}$$

On est plus ou moins en train de traiter le comportement d'un barreau de métal (infiniment fin) soumis à une « température d'entrée » à gauche et en contact avec un matériau (de température nulle) à droite.

Cette analogie a deux avantages : on traite un problème connu, ce qui pourrait permettre de faire des comparaisons, et comme c'est l'équation d'un phénomène physique, on peut attendre des bonnes propriétés de continuité.

3.5. Le schéma

On utilise le schéma suivant pour (3.4) :

$$l_j^{n+1} = l_j^n + \frac{\lambda \Delta t}{2\Delta x}\left(l_{j+1}^n - 2l_j^n + l_{j-1}^n\right) + \Delta t\left(e_j^n - h l_j^n\right). \tag{3.5}$$

Conditions aux limites périodiques Il est difficile de gérer les conditions aux bords (*i.e.* les deux unités qui n'ont qu'une seule voisine). En effet le schéma ne peut pas fonctionner puisque l'une des cases appelées n'existe pas. On peut penser à fixer des longueurs nulles aux extrémités, mais les objets qui dépasseraient la frontière seraient oubliés. On suppose donc que les usines sont en cercle : ce qui sort de la dernière rentre dans la première et réciproquement. Au niveau de l'analogie thermodynamique cela revient à considérer un anneau de métal.

Une autre solution est de prendre un grand nombre d'unités, pour que l'activité n'ait lieu qu'au centre et les unités lointaines soient à vide.

3.6. Simulation numérique

On s'intéresse, avec le code *MUP.m*, au comportement dans le cas particulier suivant. Nous utilisons les équations principales, avec au départ toutes les files d'attente de même longueur, ainsi qu'une entrée constante en fonction du temps en forme de gaussienne : une usine reçoit beaucoup de caisses de l'extérieur, ses voisines un peu moins, et les usines éloignées pas du tout. On fixe les paramètres suivants [1] :

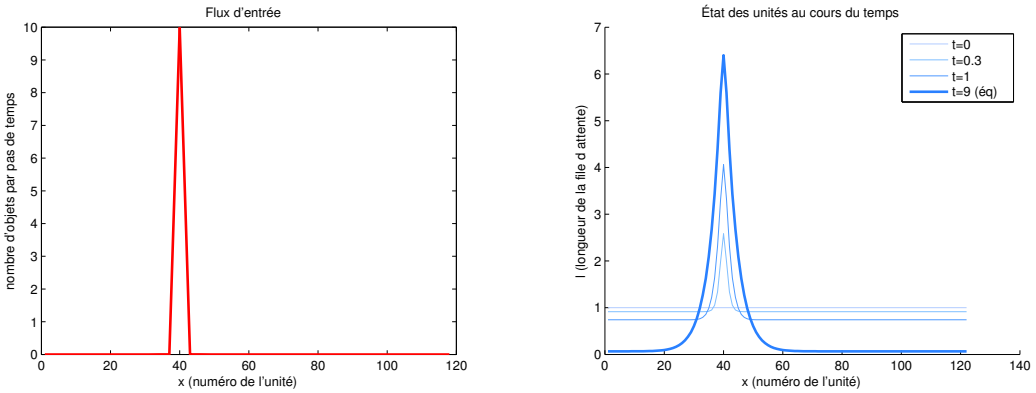
1. nombre d'usines $N = 120$;
2. paramètres du modèle : $h = 0.3$ et $\lambda = 1$;
3. paramètres de la gaussienne : $\sigma^2 = 20$ et $amplitude = 10$;
4. paramètres de discrétisation : $\Delta x = 0.1$ et $\Delta t = 0.03$.

Nous reprenons la condition de stabilité vue en section §1.2.2 en tenant compte du terme de « convection », pour obtenir la condition

$$\Delta t \leqslant \frac{1}{h + \frac{2\lambda}{\Delta x^2}} \approx 0.05,$$

qui est donc bien respectée (voir appendice C).

1. Ces paramètres sont assez arbitraires : le nombre d'usines a été choisi pour avoir des calculs assez rapides tout en conservant la pertinence de l'analogie avec un modèle continu, et les autres valeurs ont été choisies pour avoir un comportement intéressant avec une période de transition assez longue.



(a) Flux d'entrée

(b) État à différents instants

FIGURE 3.6.1.: Simulation numérique

Le résultat semble satisfaisant : les usines éloignées de la « source » ont consommé ce qu'il y avait dans leurs files, alors que les usines proches de la source ont atteint un équilibre entre l'entrée et la sortie, de sorte que les longueurs ne varient plus.

3.7. Discussion sur la valeur du modèle et conclusion sur l'approche supply chains

Le modèle précédent n'est pas réaliste pour une architecture de calcul parallèle. Cela se voit bien sûr au niveau des choix arbitraires des lois sur les flux et la sortie. Mais le problème est plus profond : en essayant d'adapter le modèle de files d'attente des supply chains, on a supposé que chaque processeur avait une liste tâches à effectuer (c'est l'analogie avec les objets à traiter), alors que le fonctionnement classique d'une architecture parallèle est synchrone.

On peut cependant penser à utiliser ce modèle pour d'autres applications. Par exemple, un réseau d'usines de traitement des eaux usées peut fonctionner plus ou moins de cette manière : l'une d'elles peut recevoir une grande quantité d'eau (par exemple lors d'un orage localisé) et se décharger sur les voisines. Quelques modifications de notre modèle permettraient alors de définir la meilleure politique.

Après la mise en lumière des ces difficultés, il semble difficile d'utiliser les outils des supply chains pour une architecture de calcul parallèle. Dans la suite on s'éloigne donc des supply chains. On propose d'abord un modèle qui utilise l'idée du *Roofline model* et qui permet une prévision du temps de calcul.

4. Modèle Roofline

Après cette étude de modèles à base de files d'attente, nous essayons de modéliser une véritable architecture de calcul parallèle ou du moins de proposer des prévisions sur les performances. Pour cela, nous étudions la thèse de Samuel Williams [Wil08], qui nous donne des informations sur une façon de prévoir les performances d'une machine.

4.1. Contexte du modèle Roofline

L'arrivée relativement récente des processeurs multi-cœurs complexifie l'évaluation des capacités des processeurs, d'où le besoin d'avoir de nouveaux modèles. Il y a plusieurs approches pour construire ces modèles, telles que l'approche statistique et l'approche *bound and bottleneck analysis* (celle du *Roofline model*, que nous présentons dans la suite).

4.2. Comportement simplifié d'un système processeur-mémoire

Le principe du système processeur-cache-DRAM est le suivant. Le processeur a besoin de données pour effectuer ses calculs, il interroge le cache. Si le cache possède cette donnée, il la transmet directement au processeur. Sinon, il la demande à la DRAM (*Dynamic Random Access Memory*), la transmet au processeur, et en conserve une copie (on parle alors de *cache miss*).

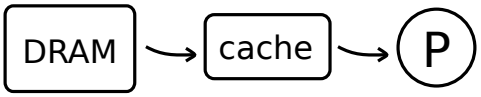


FIGURE 4.2.1.: Système processeur-cache-DRAM

La mémoire cache est beaucoup (de 10 à 100 fois) plus rapide que la mémoire DRAM (mais aussi plus petite, car plus coûteuse). Dans la suite, on néglige donc le temps d'accès au cache par rapport au temps d'accès à la DRAM. Aussi, il y a en réalité plusieurs niveaux de cache. Cela aussi est négligé dans notre modèle.

On appelle bande passante le débit maximum entre le cache et la RAM.

4.3. Le *Roofline model*

4.3.1. Caractéristiques et vocabulaire

Le *Roofline model* est un modèle relativement simple pour évaluer les performances d'une machine. Il est basé sur l'approche *bound and bottleneck analysis*, c'est-à-dire qu'il donne des limites supérieures de performance au niveau hardware. Il donne de plus l'origine de cette borne. Par exemple, sur une architecture donnée, même si on augmente le parallélisme, les performances ne vont pas augmenter car la faible bande passante empêche de dépasser un certain seuil.

Pour chaque machine, on trace un diagramme qui donne de façon graphique toute l'information nécessaire. En abscisse, l'intensité opératoire (*operationnal intensity*), c'est-à-dire le nombre d'opérations par bytes du trafic DRAM $\leftrightarrow$ cache (en flops/byte) [1], et en ordonnée le nombre d'opérations atteignable par seconde (en Gflops), donné par la formule suivante [2] (voir figure 4.3.1) :

$$\text{Gflops} = \min\{\text{Gflops(crête)}, \text{bande passante} \times \text{intensité opératoire}\}.$$

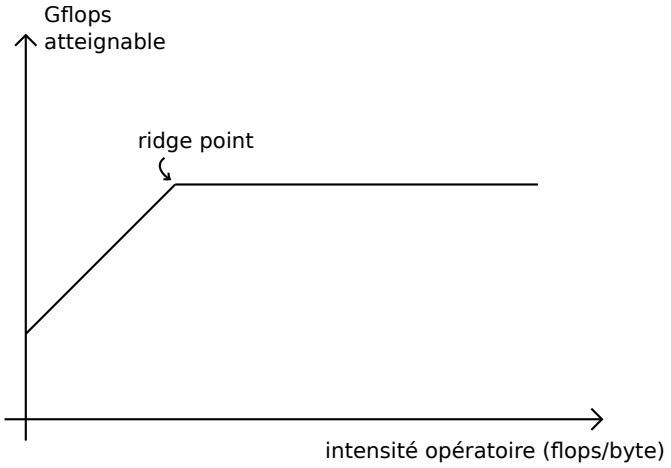


FIGURE 4.3.1.: L'origine du mot *Roofline*

Les échelles sont logarithmiques. Les courbes sont toujours affines par morceaux. Le *ridge point* est le point de cassure où la puissance maximale devient la puissance crête. Selon les choix, optimisations, etc., les segments diagonaux et horizontaux changent de place. Par exemple, un plus grand parallélisme fait monter le segment horizontal.

1. De façon informelle, l'intensité opératoire mesure si le programme que l'on exécute demande beaucoup d'accès à la mémoire, par rapport au nombre de calculs (par exemple, un petit traitement d'un gros ensemble de données), ou au contraire s'il y a peu de données demandées mais beaucoup de calculs à faire (par exemple, une simulation avec beaucoup de pas, dont on peut garder le résultat intermédiaire dans le cache : seule la donnée initiale est « chargée »).
2. Par abus de notation, on emploie les unités pour parler des grandeurs.

Pour les modélisations qui suivent, nous n'avons utilisé que les grandes lignes du *Roofline model,* cependant le détail des différentes optimisations et limites possibles pourrait être intéressant pour un travail plus approfondi, c'est pourquoi des précisions sont données en annexe (appendice B).

4.3.2. Les schémas donnés par le *Roofline model*

On fait apparaître sur le schéma différents plafonds, horizontaux et diagonaux, qui correspondent à des optimisations différentes et à plusieurs optimisations ensemble (un choix est nécessaire).

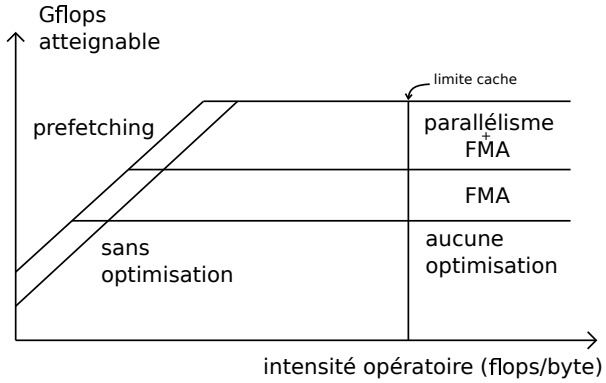


FIGURE 4.3.2.: Différents plafonds

Les *FMA* et le *prefetching* sont un peu détaillés dans l'annexe. Pour comprendre le schéma, il suffit de savoir que les premières améliorent la puissance crête, en faisant plusieurs opérations en même temps, et que le second améliore la bande passante en gérant mieux les échanges entre la RAM et le cache.

À terme, le but est d'améliorer les performances, donc d'augmenter ces plafonds. L'intérêt de ces schémas est de montrer quelles sont les améliorations les plus intéressantes. Dans le cas où l'on se place dans la portion avant le ridge point (parce que l'on fait un certain type de calcul), on aura plutôt intérêt à travailler sur les flux DRAM/cache plutôt que sur le parallélisme. Réciproquement, si l'on peut se placer à droite du *ridge point* (ce qui est mieux), on travaillera sur le parallélisme, puisque la bande passante n'est plus le facteur limitant.

5. Modélisation de la vitesse en fonction du trafic (VFT)

La thèse sur le *Roofline Model* [WP08, WWP09, Wil08] indique que la puissance de calcul peut être limitée par la bande passante si il y a beaucoup de trafic entre la DRAM et le cache, et par la puissance crête sinon. On essaye de prendre cela en compte dans le modèle suivant qui simule le comportement d'un processeur pour donner une prévision sur la vitesse d'exécution d'un programme donné.

5.1. Du programme au trafic

Dans un premier temps on présente une méthode pour calculer sommairement le trafic correspondant à un programme donné. On appelle trafic, la quantité de données qui transitent entre la mémoire et le cache par unité de temps.

5.1.1. Représentation du code

On ne prend pas en entrée un programme écrit dans un langage donné mais une représentation de ce programme, pour ne prendre en compte que les éléments intéressants au niveau de la bande passante.

On remarque que quand une variable est en jeu, il y a quatre cas qui dépendent de deux paramètres :

1. la variable est, ou n'est pas, dans le cache à cette ligne ;
2. on ne fait que lire la valeur, ou bien on la modifie [1].

Selon les cas, il y a ou non, un trafic entre le cache et la mémoire RAM (le seul trafic que l'on considère) :

	Variable dans le cache	Variable hors cache
Lecture	pas de trafic	trafic
Modification	trafic	trafic

On représente alors le code par une matrice à deux lignes : sur la première ligne la suite des variables utilisées, et sur la deuxième l'opération associée (par convention 0 pour une lecture et 1 pour une modification).

1. Par « modification » on entend définition, affectation et ré-affectation.

5.1.2. Principe de l'outil

On modélise ce qui se passe au niveau du cache pendant l'exécution du programme. On choisit d'abord une taille pour le cache, et de gérer celui-ci comme une file (structure FIFO) [2]. On conserve le trafic comme un vecteur de booléens de même taille que le vecteur qui représente le programme. Le booléen d'une case est à Vrai si et seulement si il y a un échange entre le cache et la RAM. Puis, pour chaque nouvelle variable :

- si la variable est déjà dans le cache, on la déplace en tête de la file ;
- sinon, on la rajoute en tête, ce qui fait disparaître la dernière variable du cache.

Enfin :

- si la variable considérée est dans le cache et si l'on fait une lecture, on passe le booléen correspondant à Vrai ;
- sinon, le booléen reste à Faux.

Un exemple complet est donné en appendice D.

5.2. Du trafic au temps d'exécution

On considère maintenant une grandeur y qui indique où l'on en est dans le programme (par exemple la ligne de code, si l'on ne revient pas en arrière) [3]. y dépend du temps et est strictement croissante.

L'équation de la vitesse en fonction du trafic

Maintenant, on suppose connu $T(y)$, le trafic nécessaire au point y du programme. Alors on peut proposer une équation de la forme :

$$y'(t) = d(T(y(t))),$$

où d est une fonction décroissante : plus le trafic est fort, moins la vitesse de traitement est élevée, donc moins vite le curseur se déplace.

Il nous faut choisir une fonction d qui soit pertinente pour modéliser un processeur, comme celle représentée en figure 5.2.1.

En effet, le modèle Roofline dit que jusqu'à un certain « trafic limite » c'est la puissance crête qui est limitante et le trafic n'est pas important. Puis, c'est le contraire, et plus le trafic est intense moins la vitesse de traitement est élevée. D'où le plateau de départ, puis la descente. Pour le second morceau, on propose de prendre une hyperbole. En effet, il paraît pertinent d'avoir (après le plateau) la propriété : « si le trafic est deux fois plus élevé, alors la vitesse est deux fois moins grande ».

2. On pourrait choisir d'autres politiques pour la gestion du cache, notamment à partir des fréquences d'utilisation.

3. Comme un « curseur » sur une partition de musique.

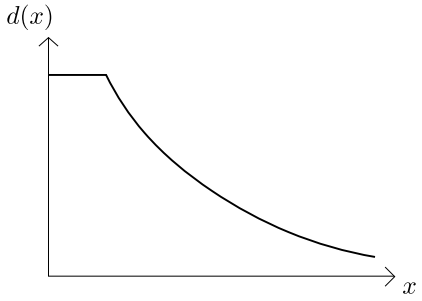
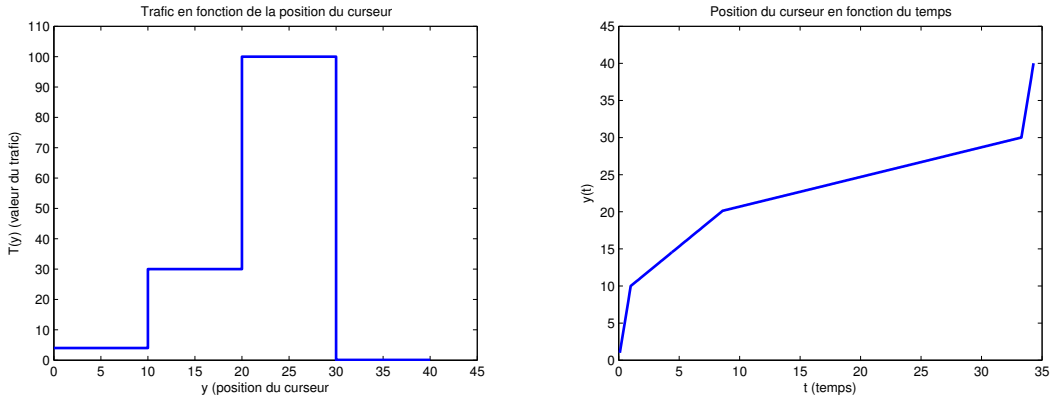


FIGURE 5.2.1.: Choix de d

5.3. Simulation numérique

Ci-dessous un exemple de l'avancement du programme (y) à partir d'un trafic donné (courbe de gauche). Comme prévu, l'avancement dans le programme est beaucoup plus lent quand un grand trafic est nécessaire.



(a) Entrée

(b) Sortie

FIGURE 5.3.1.: Calcul en fonction du trafic

5.4. Conclusion et améliorations possibles

On peut donc à partir d'une matrice représentant le code, avoir le trafic correspondant puis en déduire le temps d'exécution. Remarquons que l'on ne peut pas faire ceci dans le cas général en partant directement du code, car la terminaison même d'un programme est un problème indécidable.

6. Estimation de caractéristiques de processeurs

On utilise dans la partie précédente le point central du modèle Roofline, *i.e.* la double origine des limites de performance des processeurs : d'une part la puissance crête, d'autre part la bande passante. Dans cette partie, on essaye d'obtenir des ordres de grandeur permettant de juger la valeur de cette distinction. Plus précisément, on essaye d'avoir une idée du nombre pertinent de processeurs pour une bande passante donnée, pour pouvoir comparer avec les architectures actuelles de calcul haute performance.

6.1. Présentation du calcul

On s'inspire fortement ici de la note [DeV12] de F. De Vuyst qui est donnée en appendice E. On remplace le calcul de différences finies structuré par une simulation de mouvements de particules. Remarquons que ces deux calculs sont considérés comme importants du point de vue du calcul haute performance : ils font partie des *seven dwarfs* définis par P. Collela [Col04], les sept grands types de calcul sur lesquels on peut évaluer une architecture.

Le calcul des mouvements d'un ensemble de n particules se fait par pas de temps successifs, en actualisant les positions de toutes les particules à chaque tour de boucle. Pour cela, on doit calculer les forces qui s'appliquent sur chacune, et en supposant que l'on ne considère qu'une force entre les particules qui dépend de la distance, on a déjà n^2 distances à calculer. Cependant certaines sont moins importantes : les particules lointaines ont peu d'action individuellement, elles peuvent être regroupées. C'est l'idée de la simulation de Barnes-Hut [BH86], qui permet grâce à une structure d'arbre de faire un calcul approché avec une complexité $\mathcal{O}(n \ln(n))$. On présente ici un calcul de performance sur une version simplifiée de ce calcul.

6.2. Calcul littéral

On considère que l'espace d'étude est un cube divisé en N^3 cubes élémentaires. On suppose que l'on a M^3 processeurs qui gèrent chacun $P^3 = \left(\frac{N}{M}\right)^3$ cubes élémentaires (*i.e.* les particules présentes dans ces cubes).

Dans un premier temps on a besoin, pour chaque cube, de connaître les positions des particules situées dans les cubes adjacents. Pour les calculs d'un processeur, il faut donc connaître les positions des particules des cubes qui sont adjacents au secteur géré par le

processeur. Le nombre de tels cubes est :

$$R = (P+2)^3 - P^3 = 6P^2 + 12P + 8.$$

On suppose que l'on a n particules et qu'elles sont plus ou moins équiréparties, de telles sortes que la densité par cube est $d = \frac{n}{N^3}$. Pour chacune d'elles, il faut communiquer les trois coordonnées de position, plus le poids. On considère que toutes ces informations sont stockées sur 8 octets. D'où la communication totale par processeur pour les particules voisines (en octets) :

$$C_N = R \times d \times 4 \times 8 = 32Rd.$$

Les particules lointaines ne doivent pas être oubliées, on prend donc en compte les centres de gravité des autres processeurs [1]. Ce qui donne pour chaque processeur une communication supplémentaire en bits de :

$$C_F = 32M^3.$$

Le volume total de communication est alors :

$$C = M^3(C_N + C_F) = 32M^3(Rd + M^3).$$

Passons maintenant au nombre de calculs. Les calculs de centre de gravité sont négligeables devant les calculs des interactions entre particules. Pour un cube, le nombre d'interactions à calculer est :

$$3^3 d^2 = 27\frac{n^2}{N^6}.$$

Si chaque calcul se fait en r flops, on a le nombre de calculs par processeur :

$$I = 27\frac{n^2}{N^6} \times r \left(\frac{N}{M}\right)^3 = 27\frac{n^2}{M^3N^3}r.$$

Avant de passer à des ordres de grandeurs, on peut calculer l'intensité arithmétique :

$$a = \frac{I}{C} = \frac{27n^2r}{N^3M^6(Rd + M^3)}.$$

6.3. Ordres de grandeur

On prend les valeurs suivantes, courantes ou issues de [DeV12] :
- la performance par processeur $p = 10^{10}$ flops, et la bande passante totale $\Phi = 10^{10}$ bit/sec ;
- le nombre d'opérations pour une interaction : $r = 10$;
- $n = 10^7$, $N = 10^2$, donc $d = 10$.

1. Remarquons que les particules proches ne doivent pas être comptées deux fois. Un petit calcul est donc nécéssaire pour avoir les centres de gravité des processeurs « adjacents » auquels on a enlevé les particules comptées individuellement.

On cherche à savoir combien de processeurs sont nécessaires si l'on souhaite que les temps caractéristiques de calcul et de communications soient égaux. On calcule donc M.

On a pour le temps de communication, avec une bande passante (en bits/sec) Φ :

$$\tau_C = \frac{8C}{\Phi},$$

et pour le temps de calcul, avec une performance p :

$$\tau_I = \frac{I}{p}.$$

Alors on considère $\tau_C = \tau_I$, *i.e.* $\frac{8C}{\Phi} = \frac{I}{p}$:

$$\frac{1}{\Phi} 256 M^3 (Rd + M^3) = \frac{27 n^2 r}{N^3 M^3 p}$$
$$\frac{1}{\Phi} 256 \left[\left(6N^2 M^4 + 12 N M^5 + 8 M^6 \right) d + M^9 \right] = \frac{27 d^2 N^3 r}{p}.$$

Numériquement, on obtient $M \approx 6$.

6.4. Résultats et conclusion

Le nombre de processeurs est finalement de l'ordre de 200 ($\approx 6^3$). Le calcul de F. De Vuyst [DeV12] termine sur un résultat comparable. La conclusion est la même : aujourd'hui, les systèmes de calcul haute performance comptent de plus en plus de processeurs (bien plus que 200), la bande passante est donc un problème central, et le modèle Roofline est donc pertinent pour l'évaluation des performances.

Conclusion et perspectives

Les conclusions de ce stage sont les suivantes. Dans un premier temps, un schéma élémentaire de files d'attente, n'utilisant aucune notion probabiliste dont relève généralement cette théorie (*Queuing theory*), permet d'obtenir assez facilement une modélisation simple et pertinente des supply chains. Cependant, il semble que l'application de ce schéma à la modélisation d'une architecture de calcul parallèle (chapitre 3) ne soit pas suffisante pour en obtenir une description précise et réaliste. Dans une deuxième temps, une analyse sommaire des ordres de grandeur de temps de calcul et de bande passante (6) va dans le sens de la thèse sur le *Roofline model* [WP08, Wil08]. Nous avons aussi établi un modèle de prévision de la vitesse d'exécution d'un programme (5) qui gagnerait à être amélioré.

Bibliographie

[ADR06] D. Armbruster, P. Degond et C. Ringhofer : A model for the dynamics of large queuing networks and supply chains. *SIAM Journal of Applied Mathematics*, 66(3):896–920, 2006.

[BH86] J. Barnes et P. Hut : A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324 (4):446–449, December 1986.

[Col04] P. Collela : Defining software requirements for scientific computing. 2004.

[DeV12] Florian DeVuyst : Modèle statique grossier de type roofline pour un cluster hpc sur code différences finies 3d structuré. mai 2012.

[DGHP10] Ciro D'Apice, Simone Göttlich, Michael Herty et Benedetto Piccoli : *Modeling, Simulation, and Optimization of Supply Chains - A Continuous Approach*. SIAM, 2010.

[Wil08] S. Williams : Auto-tuning Performance on Multicore Computers. Mémoire de D.E.A., EECS Department, University of California, Berkeley, Berkeley, CA, December 2008.

[WP08] Samuel Webb Williams et David Patterson : The roofline model : A pedagogical tool for program analysis and optimization, 2008. Presentation at ParLab Summer Retreat.

[WWP09] Samuel Williams, Andrew Waterman et David Patterson : Roofline : an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, avril 2009.

A. Démonstration du théorème 2.3.2

On reprend ici la démonstration du théorème 2.3.2 en section §2.3, rappelé ci-dessous, et énoncé initialement par [DGHP10].

Théorème. *Si les délais $\Delta_m\tau(n,m)$ et $T(x_m)$ sont bornés (dans ce cas, $h_m = \mathcal{O}(\varepsilon)$), alors pour $\varepsilon \to 0$, les fonctions u_2, f_2 vérifient :*

$$\partial_t u_2(t,x) = f_2(t,x), 0 < x < X,\ t > \tau_2^I(x) \tag{A.1a}$$

$$u_2(\tau_2^I(x),x) = 0, \qquad \lim_{x\to 0^+} u_2(t,x) = \int_{\tau_2(0,0)}^{t} f^A(s)\mathrm{d}s. \tag{A.1b}$$

faiblement en x et t.

Démonstration. 1) On étend d'abord la définition des variables ρ, f, u et τ sur la droite réelle, ceci pour éviter de traiter les problèmes au bord. On pose :

$$x_m = \begin{cases} -mh_0 & m < 0 \\ X + (m-M)h_{M-1} & m > M \end{cases},$$

$$\tau^I(m) = \begin{cases} \tau^I(0) + mh_0 & m < 0 \\ \tau^I(M) + (m-M)h_{M-1} & m > M \end{cases},$$

$$\tau(n+1,m) = \begin{cases} \tau(n,m) + \varepsilon\Delta_n\tau^A(n) & m < 0,\ n \geqslant 0 \\ \tau(n,m) + \varepsilon\Delta_n\tau(n,M) & m > M,\ n \geqslant 0 \end{cases}.$$

Les égalités (2.6a) et (2.19) s'étendent donc, pour $m < 0,\ n \geqslant 0$, en

$$f(\tau(n,m),x_m) = \frac{1}{\Delta_n\tau(n,m)} = \frac{1}{\Delta_n\tau^A(n)} = f^A(\tau^A(n)).$$

D'où :

$$\lim_{x\to 0^-} f_2(t,x) = f^A(t).$$

2) Soit $\psi(t,x)$ une fonction test $\mathcal{C}^\infty$ à support compact. On considère aussi la primitive Ψ donnée par

$$\Psi(t,x_{m+1}) - \Psi(t,x_m) = h_m\psi(t,x_m), \qquad \Psi(t,x_m) = \sum_{m'=-\infty}^{m-1} h_{m'}\psi(t,x_{m'}).$$

Puisque ψ est à support compact, $\psi(\tau(n,m),x_m)$ et $\Psi(\tau(n,m),x_m)$ sont non nuls pour un nombre fini de m. D'où, pour tout $n \geqslant 0$,

$$0 = \sum_{m=-\infty}^{\infty} \left[\Psi(\tau(n,m+1),x_{m+1}) - \Psi(\tau(n,m),x_m)\right],$$

et donc

$$\begin{aligned} 0 &= \varepsilon \sum_n \sum_{m=-\infty}^{\infty} \left[\Psi(\tau(n,m+1),x_{m+1}) - \Psi(\tau(n,m),x_m)\right] \\ &= A - B, \end{aligned}$$

où l'on fait apparaître les différences en espace A et en temps B suivantes :

$$\begin{aligned} A &= \varepsilon \sum_n \sum_{m=-\infty}^{\infty} \left[\Psi(\tau(n,m+1),x_{m+1}) - \Psi(\tau(n,m+1),x_m)\right] && \text{(A.2a)} \\ &= \varepsilon \sum_n \sum_{m=-\infty}^{\infty} h_m \psi(\tau(n,m+1),x_m), \\ B &= \varepsilon \sum_n \sum_{m=-\infty}^{\infty} \left[\Psi(\tau(n,m+1),x_m) - \Psi(\tau(n,m),x_m)\right]. && \text{(A.2b)} \end{aligned}$$

a. Estimons d'abord A. Grâce à (2.12), (2.24a) et (2.26a), nous avons :

$$\begin{aligned} \int_{\tau(n,m+1)}^{\tau(n+1,m+1)} f_1(t,x_{m+1})\mathrm{d}t &= \varepsilon \Delta_n \tau(n,m+1) f(\tau(n,m+1),x_{m+1}) \\ &= \varepsilon, \\ \int_{x_m}^{x_{m+1}} f_2(t,x)\mathrm{d}x &= h_m f_1(t,x_{m+1}). \end{aligned}$$

Il suffit maintenant de remarquer que, compte tenu des hypothèses faites,

$$\sum_n \sum_{m=-\infty}^{\infty} \int_{\tau(n,m+1)}^{\tau(n+1,m+1)} \left[\int_{x_m}^{x_{m+1}} (\psi(\tau(n,m+1),x_m) - \psi(t,x)) f_2(t,x)\mathrm{d}x\right] \mathrm{d}t$$

est de l'ordre de ε. (A.2a) peut alors se réécrire :

$$\begin{aligned}
A &= \sum_n \sum_{m=-\infty}^{\infty} h_m \psi(\tau(n,m+1),x_m) \int_{\tau(n,m+1)}^{\tau(n+1,m+1)} f_1(t,x_{m+1})\mathrm{d}t \\
&= \sum_n \sum_{m=-\infty}^{\infty} \int_{\tau(n,m+1)}^{\tau(n+1,m+1)} \left[\int_{x_m}^{x_{m+1}} \psi(\tau(n,m+1),x_m) f_2(t,x)\mathrm{d}x\right] \mathrm{d}t \\
&= \sum_n \sum_{m=-\infty}^{\infty} \int_{\tau(n,m+1)}^{\tau(n+1,m+1)} \left[\int_{x_m}^{x_{m+1}} \psi(t,x) f_2(t,x)\mathrm{d}x\right] \mathrm{d}t + \mathcal{O}(\varepsilon) \\
&= \sum_{m=-\infty}^{\infty} \int_{-\infty}^{\infty} H(t-\tau^I(m+1)) \left[\int_{x_m}^{x_{m+1}} \psi(t,x) f_2(t,x)\mathrm{d}x\right] \mathrm{d}t + \mathcal{O}(\varepsilon) \\
&= \sum_{m=-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{x_m}^{x_{m+1}} H(t-\tau_2^I(x))\mathrm{d}x\mathrm{d}t + \mathcal{O}(\varepsilon) \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} H(t-\tau_2^I(x))\psi(t,x) f_2(t,x)\mathrm{d}x\mathrm{d}t + \mathcal{O}(\varepsilon) \qquad \text{(A.3)}
\end{aligned}$$

(où on a utilisé (2.26b) : $\tau_2^I(x) = \tau^I(m+1), \quad x_m \leqslant x < x_{m+1}$).

b. Traitons maintenant B. Par la formule de Taylor-Lagrange, et puisque les $\Delta_m\tau(n,m)$ sont bornés,

$$\Psi(\tau(n,m+1),x_m) - \Psi(\tau(n,m),x_m) = -\partial_t\Psi(\tau(n,m+1),x_m)\varepsilon\Delta_m\tau(n,m) + \mathcal{O}(\varepsilon).$$

Donc l'équation (A.2b) devient :

$$\begin{aligned}
B &= -\varepsilon^2 \sum_{n\geqslant 0} \sum_{m=-\infty}^{\infty} \partial_t\Psi(\tau(n,m+1),x_m)\Delta_m\tau(n,m) + \mathcal{O}(\varepsilon) \\
&= -\varepsilon^2 \sum_{n\geqslant 1} \sum_{m=-\infty}^{\infty} \partial_t\Psi(\tau(n,m+1),x_m)\Delta_m\tau(n,m) + \mathcal{O}(\varepsilon)
\end{aligned}$$

(car $-\varepsilon^2 \sum_{m=-\infty}^{\infty} \partial_t\Psi(\tau^I(m+1),x_m)\Delta_m\tau^I(m) = \mathcal{O}(\varepsilon)$). Or, d'après (2.20), nous pouvons réécrire $\Delta_m\tau(n,m)$:

$$\Delta_m\tau(n,m) = \frac{h_m}{\varepsilon X}\rho(\tau(n-1,m+1),x_m)\Delta_n\tau(n-1,m+1),$$

d'où :

$$B = -\varepsilon \sum_{n\geqslant 1} \sum_{m=-\infty}^{\infty} \frac{h_m}{X}\partial_t\Psi(\tau(n,m+1),x_m)\rho(\tau(n-1,m+1),x_m)\Delta_n\tau(n-1,m+1) + \mathcal{O}(\varepsilon).$$

Maintenant, on répète essentiellement la même technique utilisée pour le terme A. D'après (2.24b),

$$\int_{\tau(n-1,m+1)}^{\tau(n,m+1)} \rho_1(t,x_m)\mathrm{d}t = \varepsilon\Delta_n\tau(n-1,m+1)\rho(\tau(n-1,m+1),x_m),$$

donc

$$B = -\sum_{n=1}^{\infty}\sum_{m=-\infty}^{\infty} \frac{h_m}{X}\partial_t\Psi(\tau(n,m+1),x_m)\int_{\tau(n-1,m+1)}^{\tau(n,m+1)} \rho_1(t,x_m)\mathrm{d}t + \mathcal{O}(\varepsilon)$$
$$= -\sum_{m=-\infty}^{\infty} \frac{h_m}{X}\int_{\tau^I(m+1)}^{\infty} \partial_t\Psi(t,x_m)\rho_1(t,x_m)\mathrm{d}t + \mathcal{O}(\varepsilon)$$

(là aussi, $\sum_{n\geqslant 1}\int_{\tau(n-1,m+1)}^{\tau(n,m+1)}\left[\partial_t\Psi(\tau(n,m+1),x_m) - \partial_t\Psi(t,x_m)\right]\rho_1(t,x_m)\mathrm{d}t = \mathcal{O}(\varepsilon)$). On transforme également $\frac{h_m}{X}\rho_1(t,x_m)$ grâce à (2.25b) :

$$B = -\sum_{m=-\infty}^{\infty}\int_{-\infty}^{\infty} H(t-\tau^I(m+1))\partial_t\Psi(t,x_m)[u_1(t,x_{m+1}) - u_1(t,x_m)]\mathrm{d}t + \mathcal{O}(\varepsilon).$$

Ici, une petite sommation d'Abel donne :

$$\begin{aligned}
B &= -\sum_{m=-\infty}^{\infty}\int_{-\infty}^{\infty} H(t-\tau^I(m))\partial_t[\Psi(t,x_m) - \Psi(t,x_{m-1})]u_1(t,x_m)\mathrm{d}t + \mathcal{O}(\varepsilon)\\
&= -\sum_{m=-\infty}^{\infty}\int_{-\infty}^{\infty} H(t-\tau^I(m))h_{m-1}\partial_t\psi(t,x_{m-1})u_1(t,x_m)\mathrm{d}t + \mathcal{O}(\varepsilon)\\
&= -\sum_{m=-\infty}^{\infty}\int_{-\infty}^{\infty} \partial_t\psi(t,x_{m-1})\left[\int_{x_{m-1}}^{x_m} H(t-\tau_2^I(x))u_2(t,x)\mathrm{d}x\right]\mathrm{d}t + \mathcal{O}(\varepsilon)\\
&= -\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \partial_t\psi(t,x)H(t-\tau_2^I(x))u_2(t,x)\mathrm{d}x\mathrm{d}t + \mathcal{O}(\varepsilon) \qquad \text{(A.4)}
\end{aligned}$$

(en commettant encore une erreur de l'ordre de ε à la dernière étape).

3) En combinant (A.3) et (A.4), on a donc, pour toute fonction test ψ,

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(t-\tau_2^I(x))\left[\psi(t,x)f_2(t,x) + \partial_t\psi(t,x)u_2(t,x)\right]\mathrm{d}x\mathrm{d}t = \mathcal{O}(\varepsilon),$$

c'est-à-dire,

$$\int_{-\infty}^{\infty}\left([\psi(t,x)u_2(t,x)]_{\tau_2^I(x)}^{\infty} + \int_{\tau_2^I(x)}^{\infty}\psi(t,x)\left[f_2(t,x) - \partial_t u_2(t,x)\right]\mathrm{d}t\right)\mathrm{d}x = \mathcal{O}(\varepsilon),$$

ce qui, lorsque $\varepsilon \to 0$, est la formulation faible de

$$\partial_t u_2(t,x) = f_2(t,x),\ x\in\mathbb{R},\ t > \tau_2(x), \quad u_2(\tau_2^I(x),x) = 0.$$

Par définition de $u_1(t,x_0)$ en (2.25a), on a aussi :

$$\lim_{x\to 0^+} u_2(t,x) = \int_{\tau_2(0,0)}^{t} f^A(s)\mathrm{d}s.$$

□

B. Précisions sur le Roofline model

On détaille ici les différentes limites techniques à la performance présentées par S. Williams dans [Wil08]. Certaines sont purement techniques et ne peuvent pas être introduites dans un modèle mathématique, mais d'autres comme le parallélisme pourraient être developpées, d'où l'intérêt de cette section.

On donne aussi une loi classique sur le parallélisme (la *loi d'Amdhal*), qui peut permettre de quantifier ce parallélisme.

B.1. Limites de performance

B.1.1. Les limites au niveau des méthodes de calcul

Les méthodes de calcul changent les plafonds horizontaux : la puissance crête atteignable. Les deux principaux changements possibles sont au niveau du parallélisme et de la fusion multiplication-addition. Plus le calcul est parallélisé plus la puissance crête augmente. En effet, celle-ci est mesurée en nombre de calculs flottants par unité de temps. On l'augmente en faisant plus de calculs par tour d'horloge. Un autre aspect, moins évident, est la fusion des opérations : les processeurs modernes peuvent effectuer en un temps, l'opération FMA (fused multiply-add) : $a \leftarrow a + b \times c$, en utilisant cette fusion on augmente la puissance crête (ou, selon le point de vue, on ne la diminue pas : les performances annoncées par les constructeurs sont calculées en supposant que toutes les opérations sont des FMA, faire une simple addition est alors synonyme d'une perte de temps) [1].

B.1.2. Les limites au niveau de la mémoire

En augmentant la bande passante, on repousse le plafond de la courbe avant le *ridge point* (segment oblique) car on augmente le coefficient directeur de la droite et la représentation est log-log. Il y a trois principales méthodes pour augmenter la bande passante :

- restructurer le code pour avoir le moins de trafic possible entre la DRAM et les caches (notamment en regroupant les informations par paquets cohérents) ;
- améliorer l'« affinité mémoire » (si il y a plusieurs processeurs, faire en sorte que des tâches associées à un même ensemble de données soient toutes effectuées par le même processeur, pour éviter du trafic inutile) ;
- faire un bon *prefetching* : charger les données futures avant d'en avoir besoin (pour ne pas avoir à les attendre quand il faudra les traiter).

1. Cette opération peut sembler bizarre, mais dans un produit matriciel par exemple, cela divise par deux le temps de calcul car on ne fait qu'additionner des produits de deux éléments.

B.1.3. Les limites au niveau des caches (3C model)

On a parlé des courbes et des plafonds au niveau de la puissance atteignable, en fait l'intensité opératoire est elle aussi bornée, on peut donc tracer une droite verticale qui donne l'intensité opératoire maximale (*Arithmetic Intensity Wall*). Celle-ci est fixée par trois paramètres principalement (c'est le modèle des *3Cs of cache*) qui décrivent les différents *cache misses* possibles :

- les *compulsory misses* (certains accès DRAM/cache sont obligatoires pour aller récupérer les données au premier accès) ;
- les *capacity misses* (les données nécéssaires au programme peuvent être plus grosses que l'espace disponible dans le cache) ;
- les *conflict misses* (les problèmes dans la gestion des informations dans le cache).

B.2. Exemple d'un autre outil de prévision de performance

Une autre approche du parallélisme et des améliorations est la *loi d'Amdhal* (semi-empirique), qui donne le gain en performance pour une amélioration. Soient T le temps d'exécution sans l'amélioration, T_a le temps avec l'amélioration, s la fraction du temps concernée par l'amélioration [2], et A_c l'accélération. On a alors :

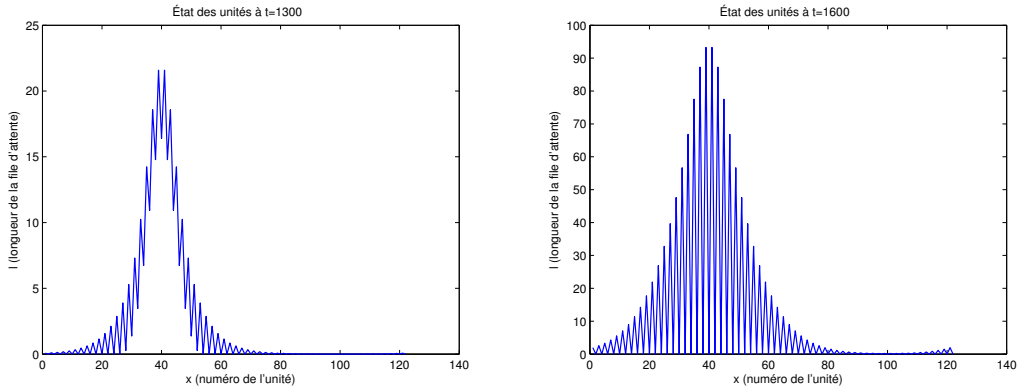
$$T_a = (1 - s)T + \frac{sT}{A_c}.$$

2. On peut, par exemple, diviser par deux le temps d'exécution d'une partie d'un programme en la parallélisant alors que les autres parties ne peuvent pas être parallélisées.

C. Calcul MUP : étude sommaire de stabilité

On a utilisé pour MUP, les pas de temps $\Delta t = 0.03$ et d'espace $\Delta x = 0.1$. Ceux-ci assurent une stabilité L^∞. Nous avons aussi essayé avec d'autres pas de temps. Une petite étude (assez experimentale) des résultats est donnée ici.

En reprenant les paramètres donnés en (3.6) avec $\Delta t = 0.1$ plutôt que $\Delta t = 0.03$, on observe, à partir de $t = 1000$ la singularité suivante, alors que les premiers instants ne laissaient présager aucune instabilité.



(a) $t = 1300$ (b) $t = 1600$

FIGURE C.0.1.: Instabilité pour $\Delta t = 0.1$

Remarquons que l'on a deux sortes de divergence : la norme infinie grandit, mais ce que l'on voit surtout c'est une instabilité « oscillante ».

Étude de la dérive Pour étudier cette dérive, on définit un indice de stabilité qui dépend du pas m, et que l'on note $is(m)$:

$$is(m) = \frac{1}{N} \sum_{n=1}^{N} |l(n,m) - l(n,m+1)|.$$

Plus on est dans une situation stable, moins les valeurs varient entre deux pas, plus cet indice est faible. À la limite, si l'on reste avec exactement les mêmes valeurs, on obtient $is = 0$. On trace $\log(is)$ en fonction de m.

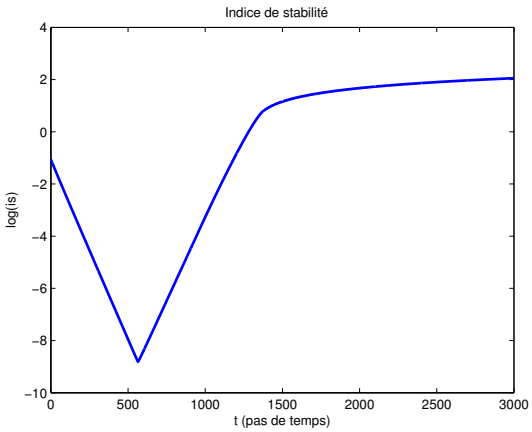


FIGURE C.0.2.: Indice de stabilité

On voit que l'indice de stabilité tend d'abord fortement vers 0 et à une vitesse exponentielle, puis la simulation « déraille » complétement (autour du $500^{ème}$ pas), et is croit de façon exponentielle.

D. Exemple d'un produit matriciel pour l'outil de VFT

On considère le produit matriciel :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

On ne s'occupe pas des résultats intermédiaires qu'on suppose stockés dans une case à part. Le résultat est un ensemble de 4 variables :

$$\begin{cases} r_1 = ae + bg \\ r_2 = af + bh \\ r_3 = ce + dg \\ r_4 = cf + dh \end{cases}$$

La matrice qui code le produit est donc la suivante (représentée sous forme de tableaux ici mais bien en matrice en matlab) :

r_1	a	e	b	g	r_2	a	f	b	h	r_3	c	e	d	g	r_4	c	f	d	h
1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0

On numérote les variables (en nommant d'abord les variables lues). D'où :

9	1	5	2	7	10	1	6	2	8	11	3	5	4	7	12	3	6	4	8
1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0

et le « parcours » suivant (la tête du cache est à gauche) :

variable	cache								trafic
9	9	0	0	0	0	0	0	0	1
...									
10	10	7	2	5	1	9	0	0	1
1	1	10	7	2	5	9	0	0	0
...									
8	8	2	6	1	10	7	5	9	1
11	11	8	2	6	1	10	7	5	1
...									

(Le tableau étant relativement long on a enlevé certaines parties.) On peut remarquer plusieurs choses.

1. Au départ le cache est vide, ce qui est représenté par des 0.
2. À chaque fois que l'on s'intéresse à une variable qui n'a jamais été utilisée le trafic est de 1, ce qui est le cas notamment au début, pour les variables 9, 1, 5, 2, 7 et 10.
3. Lorsque le 1 réapparait il est toujours dans le cache et on ne fait que le lire, donc le trafic est nul et le cache subit juste une petite permutation.
4. Entre les variable 8 et 11, on peut voir sur le tableau que l'on perd une variable : puisque la variable 9 n'a pas été utilisée depuis longtemps, elle sort du cache ; si on en a de nouveau besoin il faudra aller la chercher dans la RAM (trafic = 1).

Le trafic total est 15 pour ce calcul, avec un cache de 8 cases.

On peut voir la dépendance du trafic en fonction de la taille du cache sur le tableau suivant :

taille du cache	1	2	3	4	5	6	7	8	9	10	11	12
trafic	20	20	20	20	16	16	16	15	13	12	12	12

On ne fait le tableau que jusqu'à la taille 12 pour le cache car il n'y a que 12 variables.

Comme prévu, pour une petite taille de cache, on a autant d'accès DRAM que d'utilisations de variables, et pour un cache grand, il y a autant de trafic que de définitions de variables.

E. Modèle statique grossier de type Roofline pour un cluster HPC sur code volumes finis 3D structurés

(Note de F. De Vuyst.)

E.1. Cadre et hypothèses

Le cadre de calcul HPC ici est celui d'un code 3D structuré de différences finies sur des opérateurs aux dérivées partielles locaux. On suppose les schémas numériques explicites. Le cadre est par exemple celui d'un système d'équations hyperboliques non linéaires.

1. Supposons une discrétisation uniforme sur un cube avec N points de discrétisation par direction. On a donc N^3 points de discrétisation.
2. Supposons une parallélisation massive du code par une décomposition de domaines en M sous-domaines par directions, soit M^3 sous-domaines au total. On suppose que le nombre de processeurs est le nombre de sous-domaines M^3. Le nombre de points par sous-domaine est donc $(N/M)^3$. Voir figure E.1.1.
3. On suppose qu'il y a V variables de calcul par maille, le nombre total de degrés de libertés est donc $V\,N^3$.
4. Les variables sont toutes des flottants de type `double` codés sur 8 octets, donc un stockage total de $(8VN^3)$ octets.
5. La taille de stockage par sous-domaine (et donc par processeur) est de $8V(N/M)^3$.
6. On suppose des conditions aux limites périodiques. Chaque sous-domaine (processeur) communique avec des 6 voisins (processeurs) à chaque pas de temps Δt. Chaque interface possède $(N/M)^2$ mailles. Pour des raisons de reconstruction de gradients par exemple, on suppose que chaque processeur communique 2 couches de faces, soit $2(N/M)^2$ par face. En considérant les 6 faces, les V variables, et les représentations des doubles sur 8 octets, on a ainsi $96V(N/M)^2$ d'octets à communiquer par sous-domaines. En considérant les M^3 sous-domaines, on a donc un volume de communications en octet de
$$96V(N/M)^2M^3 = 96VN^2M.$$
7. On suppose être seul utilisateur sur le cluster HPC, si bien qu'il n'y a pas de charge ni de trafic exogènes.

8. Soit p la performance par processeur, en flop/sec (opération flottante par seconde). On note a l'intensité arithmétique (en flop/octet). Pour un code de différences finies explicites, a est un $O(1)$. Le temps caractéristique de calcul sur un processeur est ainsi égal à (en sec) :
$$\tau_c = \frac{8aV(N/M)^3}{p}.$$

9. On note Φ la bande passante totale du cluster (en bit par seconde). On suppose qu'il n'y a pas de réseau propriétaire ou en organisation hypercube, un seul réseau sert à la communication. La bande passante en octet est donc égale à $\Phi/8$ et le temps caractéristique de transfert de toutes les données par pas de temps est égal à
$$\tau_t = \frac{8(96V(N/M)^2M^3)}{\Phi} = \frac{768VN^2M}{\Phi}.$$

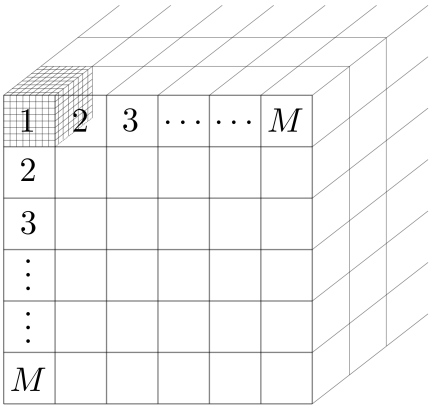


FIGURE E.1.1.: Organisation du calcul parallèle et décomposition de domaine

E.2. Analyse des performances

A titre introductif, considérons le cas où il y a équilibre entre temps de calcul et temps de communication, soit $\tau_c = \tau_t$. Dans ce cas, le temps de transfert réduit par deux le temps idéal de calcul sans communications, ce qui correspond à une efficacité de parallélisation égal à 1/2. L'équation $\tau_c = \tau_t$ donne
$$\frac{8aV(N/M)^3}{p} = \frac{768VN^2M}{\Phi},$$
c'est-à-dire
$$\frac{96M^4}{\Phi} = \frac{aN}{p}$$
Pour approfondir l'étude, il faudrait généraliser en introduisant les notions de speedup et d'efficacité.

E.3. Scenario 1 : calcul du nombre de processeurs à fréquence et bande passante données

Considérons le cas ci-dessus $\tau_c = \tau_t$. Pour obtenir une efficacité de 1/2, à p et Φ données, on obtient

$$M = \left(\frac{aN\Phi}{96p}\right)^{1/4}.$$

Remarque : le résultat est indépendant du nombre de variables V.

E.3.1. Application numérique

Considérons le dimensionnement suivant : $a = 10$ [flop/octet], $p = 10^{10}$ [flop/sec], $N = 4096$ (calcul 4096^3), $\Phi = 10^{10}$ [bit/sec]. On obtient $M = 4.5449$. En arrondissant à M=5, on obtient un cluster de $M^3 = 125$ processeurs. Ce résultat est assez surprenant. Le facteur limitant est clairement la bande passante. L'efficacité va très vite être affectée pour un cluster constitué de plus de 125 processeurs (sous les hypothèses mentionnées ci-dessus).

Supposons $V = 20$. Cela donne les caractéristiques suivantes :

- Nb de processeurs $M^3 = 125$
- Puissance totale théorique du cluster : $P = M^3 p = 1.25\,10^{12}$ [flop/sec].
- Volume de communications par pas de temps : $C = 96VN^2M = 1.61\,10^{11}$ [bit/sec].

E.4. Scenario 2 : calcul de la bande passante cible à nb de processeurs et fréquence donnés

On obtient la formule

$$\Phi = \frac{96pM^4}{aN}$$

(soulignons que la bande passante se comporte en M^4 !).

E.4.1. Application numérique

Considérons $M = 64$, soit un cluster de $M^3 = 262144$ processeurs, $a = 10$, $p = 10^10$ [flop/sec], $N = 4096$. On obtient

$$\Phi = 3.93\,10^{14}\ [bit/sec].$$

Ce calcul montre le besoin de réseaux inter-nœuds très haut débit (en peta-bits/sec) pour exploiter pleinement la puissance de calcul du cluster.

F. Codes MATLAB

À titre d'illustration, nous plaçons ici quelques extraits de code écrit au cours de nos différentes modélisations. Le lecteur pourra retrouver et utiliser librement toutes les sources MATLAB à l'adresse : http ://www.dptinfo.ens-cachan.fr/~bdadoun/supply_chains/.

F.1. Modèles simples de files d'attente

F.1.1. Modèle discret

```
%% Paramètres du système
% Domaine temporel
dt = 0.1;
tmax = 130;
t = 0 : dt : tmax;
P = length(t);
% Nombre d'objets
N = length(t);
% Nombre de processeurs
M = 4;
% Capacités et temps de traitement
T = [1 3 1 0];
mu = [15 10 15 30];
% Flux d'entrée
f_in_pp = mkpp([0 40 60 80 130], [5; 12.5; 17; 5]);
noise_pp = mkpp(t, normrnd(0, 0.5, N - 1, 1));
f_in = @(t)(ppval(f_in_pp, t) + ppval(noise_pp, t));

%% Initialisation du calcul
tau = zeros(N, M);
U = zeros(P, M);
W = zeros(P, M - 1);
F = zeros(P - 1, M);

%% Calcul
% tau(0, *) (arrivées du premier objet)
for m = 2 : M
    tau(1, m) = tau(1, m - 1) + T(m - 1);
end
% tau(*, 0) (arrivées au premier processeur)
for n = 2 : N
    tau(n, 1) = 1/f_in(tau(n - 1, 1)) + tau(n - 1, 1);
end
% récursivité
for m = 1 : M - 1
```

```
    for n = 2 : N
        tau(n, m + 1) = max(tau(n, m) + T(m), tau(n - 1,m + 1) + 1/mu(m));
    end
end
% U
for m = 1 : M
    for p = 1 : P
        U(p, m) = length(find(tau(:, m) < t(p)));
    end
end
% W
for m = 1 : M - 1
    for p = 1 : P
        W(p, m) = U(p, m) - U(p, m + 1);
    end
end
% F
for m = 1 : M
    for p = 1 : P - 1
        F(p, m) = (U(p + 1, m) - U(p, m)) / dt;
    end
end

%% Affichage
for m = 1 : M - 1
    plot(t, W(:, m), 'LineWidth', 2);
    hold all;
end
legend(int2str([1 : M-1]'));
xlabel('t (temps)');
ylabel('W (work in progress)');
title('Quantité d''objets dans les files d''attente (cas discret)');
```

F.1.2. Modèle continu

```
%% Paramètres du système
% Domaine temporel
dt = 0.1;
tmax = 130;
t = 0 : dt : tmax;
% Domaine spatial
h = 0.02;
X = 1;
x = h : h : X;
% Nombre d'objets
N = length(t);
% Nombre de processeurs
M = length(x);
% Capacités et temps de traitement
T0 = 0.1;
T = T0 * ones(1, M);
```

```
mu = ones(N, 1) * [15 * ones(1, M / 5), 10 * ones(1, 3 * M / 5), 15 * ones
    (1, M / 5)];
% Flux d'entrée
f_in_pp = mkpp([0 40 60 80 130], [5; 12.5; 17; 5]);
noise_pp = mkpp(t, normrnd(0, 0.5, N - 1, 1));
f_in = @(t)(ppval(f_in_pp, t) + ppval(noise_pp, t));

%% Initialisation du calcul
rho = zeros(N, M);
f = zeros(N, M);

%% Calcul
% f et rho (équations (36a) et (36b))
f(:, 1) = f_in(t');
for n = 1 : N - 1
    f(n, 2 : M) = min([mu(n, 1 : M - 1); X / (M * T0) * rho(n, 1 : M - 1)])
        ;
    rho(n + 1, 1 : M - 1) = rho(n, 1 : M - 1) - dt / h * (f(n, 2 : M) - f(n
        , 1 : M - 1));
end

%% Affichage
S1 = sum(rho(:, 1 : 10)') * h;
S2 = sum(rho(:, 11 : 40)') * h;
S3 = sum(rho(:, 41 : 50)') * h;
plot(t, S1, t, S2, t, S3, 'LineWidth', 2);
legend(int2str([1; 2; 3]));
xlabel('t (temps)');
ylabel('W (work in progress)');
title('Quantité d''objets dans les files d''attente (cas continu)');
```

F.2. Modélisation d'unités en parallèle

```
%% Paramètres "physiques"
% Coefficient de diffusion
lambda = 1;
% mode de sortie
sortie_proportionnelle = 1;
% Coefficient de sortie
h = 0.3;
sortie_constante = 0;
sortie = 0.1;
% Longueur au départ
l_ini = 1;

%% Paramètres de discretisation
% pas
dx = 0.1;
dt = 0.03;
% Nombre de pas de temps
T = 3000;
```

```
% Nombre de pas d'espace
N = 120;
% +2 cases "fantômes"
Nf = N+2;
% Matrice des longueurs : l(x,t)
l = zeros(Nf,T);
% Matrice des entrées : e(x,t)
e = ones(Nf,T);

%% Entrée
centre = floor(Nf/3);
sigma_carre = 20;
amplitude = 10;
for t = 1:T
    for x=1:Nf
        e(x,t)= amplitude*exp(-((x-centre)^2)/sigma_carre);
        e(1,t) = 0;
        e(Nf,t) = 0;
    end;
end;

%% Paramètre d'affichage
affichage_apres = 1;
afficher_matrice = 0;
temps_pause = 0.1;

%% Initialisation du calcul
for x = 1:Nf
    l(x,1) = l_ini;
end;

%% Calcul
if sortie_proportionnelle
for t = 2:T
    for x = 2:(Nf-1)
        l(x,t) = l(x,t-1) + (lambda*dt/(2*dx))*(l(x+1,t-1)+l(x-1,t-1)-2*l(x
            ,t-1)) + dt*(e(x,t)-h*(l(x,t-1)));
        if l(x,t) < 0
            l(x,t)=0;
        end;
    end;
    l(1,t) = l(Nf-1,t);
    l(Nf,t) = l(2,t);
end;
end;
if sortie_constante
for t = 2:T
    for x = 2:(Nf-1)
        l(x,t) = l(x,t-1) + (lambda*dt/(2*dx))*(l(x+1,t-1)+l(x-1,t-1)-2*l(x
            ,t-1)) + dt*e(x,t)- sortie;
        if l(x,t) < 0
            l(x,t)=0;
        end;
```

```
    end;
    l(1,t) = l(Nf-1,t);
    l(Nf,t) = l(2,t);
end;
end;

%% Affichage
if afficher_matrice
    l
end
if affichage_apres
    axis([0 N 0 l_ini]);
    m = 1;
    while m < T
        plot(l(:,m));
        title('État des unités à t=1300');
        xlabel('x (numéro de l''unité)');
        ylabel('l (longueur de la file d''attente)');
        pause(temps_pause);
        m = m+1;
    end;
end;
```