RenPar'9,
Lausanne. Suisse, 20–23 mai 1997.

Calcul de la suite optimale de taille de paquets pour la factorisation de Cholesky

Pierre Ramet *
LaBRI (URA CNRS 1304), Université Bordeaux I, 33 405 Talence Cedex, France
E-mail: `ramet@labri.u-bordeaux.fr`

mots clés: Factorisation de Cholesky, taille optimale de paquets, recouvrement des communications.

1. Introduction et Problématique

L'utilisation des machines parallèles à mémoire distribuée apporte un gain important en performances et en taille mémoire mais amène en contre partie un surcoût en communications. Afin d'obtenir des programmes performants et extensibles, il convient de masquer ce surcoût. Plusieurs solutions existent. La première consiste en un choix judicieux de la distribution des données qui réduira au maximum le nombre et la taille des communications. De plus, si les dépendances le permettent, on essaiera d'initialiser les communications au plus tôt et de manière asynchrone : pendant l'exécution de ces communications, on effectuera d'autres calculs, sans rapport avec l'échange de données. Des algorithmes ont été proposés pour recouvrir les communications en particulier dans le cas des transformées de Fourier [8], D. W. Walker étant l'inventeur de la technique de recouvrement dans la conception de programmes parallèles. Dans [6], la taille optimale des paquets est calculée pour des nids de boucles DOACROSS généraux.

Notre étude se place dans le cadre des *problèmes irréguliers*. L'irrégularité entraîne, par opposition aux études précédentes, un calcul non constant de la taille optimale de paquets. Dans [5], une solution adaptative pour recouvrir les communications est proposé dans le cas du produit matrice-vecteur. Un protocole de communication est proposé dans [7] pour offrir un grain de calcul adaptatif, dans le cadre des calculateurs à mémoire partagée. Dans notre article, on présente un schéma de calcul de la suite de taille de paquets qui optimise le recouvrement pour le problème de la factorisation de Cholesky d'une matrice symétrique. La *symétrie* du problème engendre déjà une irrégularité dans le sens où la masse des calculs à traiter évolue au cours du macro-pipeline; on propose alors une méthode pour déterminer analytiquement le découpage du macro-pipeline, en tenant compte des contraintes de granularité imposées par les optimisations vectorielles de type BLAS. L'étape suivante consistera à généraliser ces résultats aux problèmes de l'algèbre linéaire creuse, l'objectif étant à terme d'intégrer ces techniques dans le cadre d'un schéma de compilation automatique. Pour cela il sera nécessaire d'avoir une analyse du code donnant une estimation fine de la complexité.

La suite de cet article est organisée de la façon suivante. La partie 2 introduit la formulation du problème et sa résolution. Dans la partie 3 nous présentons les résultats obtenus sur un SP2 IBM. Enfin, une conclusion termine cet article sur la suite de l'étude.

2. Calcul de la suite optimale

Dans cette partie, nous présentons une méthode pour calculer une suite de taille de paquets qui maximise le recouvrement pour l'algorithme de factorisation de Cholesky. Le point de départ est l'algorithme séquentiel colonne classique que l'on modifie pour mettre en évidence le macro-pipeline :

*Ce travail est soutenu par le GDR-PRC PRS

Algorithme Séquentiel :

Pour $k := 1$ à n faire
  Pour $j := k$ à n faire
    $a_{j,k} := a_{j,k}/\sqrt{a_{k,k}}$
  Pour $j := k+1$ à n faire
    Pour $i := j$ à n faire
      $a_{i,j} := a_{i,j} - a_{i,k} * a_{j,k}$

Algorithme modifié par paquet :

Pour $k := 1$ à n faire
  $a_{k,k} := \sqrt{a_{k,k}}$
  Pour $p := 1$ à P_k faire
    Pour $j := j_p$ à $j_{p+1} - 1$ faire
      $a_{j,k} := a_{j,k}/\sqrt{a_{k,k}}$ (BLAS)
    Pour $q := 1$ à $p - 1$ faire
      Pour $jj := j_q$ à $j_{q+1} - 1$ faire
        Pour $j := j_p$ à $j_{p+1} - 1$ faire
          $a_{j,jj} := a_{j,jj} - a_{j,k} * a_{jj,k}$ (BLAS)
    Pour $j := j_p$ à $j_{p+1} - 1$ faire
      Pour $i := j$ à $j_{p+1} - 1$ faire
        $a_{i,j} := a_{i,j} - a_{i,k} * a_{j,k}$ (BLAS)

La redistribution des calculs correspond à une progression "ligne" dans la factorisation. La symétrie conduit à des tailles de paquets variables au cours du pipeline. A chaque étape de la factorisation, on découpe la colonne pivot en paquets de tailles inégales; c'est cette suite de taille de paquets que l'on cherche à optimiser. Pour l'étape k, il y a P_k paquets de taille ν_p correspondant aux indices de ligne j_p à $j_{p+1}-1$ (voir figure 1). On calcule les contributions engendrées par le paquet en cours (bloc diagonal) ainsi que le couplage avec les paquets précédents. Il est donc nécessaire de conserver les paquets précédement reçus tant que le pipeline n'est pas terminé.

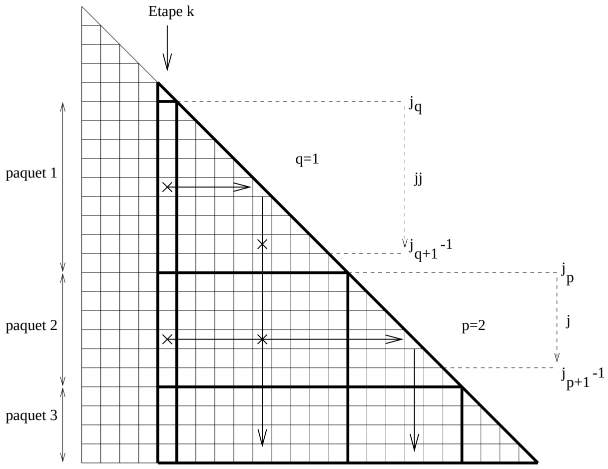

Figure 1. Factorisation de Cholesky

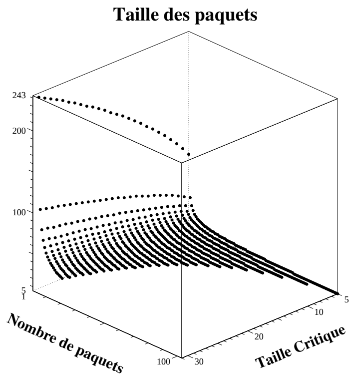

Figure 2. Suite de taille de paquets

On appelle "calcul avant" ($Tcalc_A$) le travail qui doit être effectué sur un paquet avant de communiquer les résultats correspondants. De même, le "calcul après" ($Tcalc_B$) représente le travail qui doit être effectué à l'aide des résultats communiqués. On note $\tau_\times$ le temps élémentaire de mise à jour et $\tau_\div$ le temps élémentaire du calcul de la contribution.

Le processeur émetteur effectue les mises à jour de ses colonnes dans le "calcul avant", de sorte que le "calcul avant" et le "calcul après" soient asymptotiquement équilibrés.

Pour un paquet p de taille ν_p, pour la colonne k, $1 \le k \le n$, $1 \le p \le P_k$, on a :

$$Tcalc_A(\nu_p) = \frac{\sum_{q=1}^{p-1} \nu_p \nu_q \tau_\times + \frac{\nu_p(\nu_p+1)}{2}\tau_\times}{NProcs} + \nu_p \tau_\div \quad \text{et} \quad Tcalc_B(\nu_p) = \frac{\sum_{q=1}^{p-1} \nu_p \nu_q \tau_\times + \frac{\nu_p(\nu_p+1)}{2}\tau_\times}{NProcs}.$$

En supposant que la taille des paquets est supérieure au nombre de processeurs, et grâce à une *distribution cyclique*, tous les processeurs sont concernés par les contributions et on émettra les paquets à l'aide d'un "broadcast".

Quand p croît, la taille du calcul augmente comme $p\nu_p^2$, et donc nécessairement la suite de taille de paquets est décroissante. Pour expliciter le couplage entre les temps de calcul et de communication, qui évoluent de manière opposée, on impose la contrainte :

$$\forall p,\ Tcalc_A(\nu_p) + Tcomm(\nu_p) = C,$$

avec C à déterminer (voir proposition 2). D'autre part, on considère un modèle linéaire de communication : $Tcomm(\nu_p) = \beta + \nu_p \tau$. Sous ces hypothèses, on démontre le résultat suivant :

Proposition 1. *Si L est la taille du pipeline, la suite de taille de paquets est donnée par :*

$$\left\{ \begin{array}{l} \nu_1(C) = \frac{-b+\sqrt{b^2+4a(C-\beta)}}{2a} \\ \forall p \geq 2, \quad \nu_p(C) = \frac{\sqrt{b^2+4ap(C-\beta)}-\sqrt{b^2+4a(p-1)(C-\beta)}}{2a} \end{array} \right. ,\ avec \left\{ \begin{array}{l} a = \frac{\tau_\times}{2NProcs} \\ b = \frac{\tau_\times}{2NProcs} + \tau + \tau_\div \end{array} \right. .$$

De plus le nombre de paquets du pipeline est donné par : $P_k(C) = \frac{L(b+aL)}{C-\beta}$.

L'utilisation du noyau de calcul BLAS impose une taille minimale de paquets Λ.

On cherche donc C vérifiant les contraintes : $\left\{ \begin{array}{l} \forall p, \nu_p \geq \Lambda, \\ \forall p, Tcalc(\nu_p) \geq Tcomm(\nu_p). \end{array} \right.$

On démontre le résultat suivant :

Proposition 2. $C = \max(C_1, C_2)$ *avec :*

$$\left\{ \begin{array}{ll} C_1 = \beta + 2aL\Lambda + b\Lambda - a\Lambda^2 \geq 0 & \textit{obtenue avec la } 1^{i\grave{e}re} \textit{ contrainte,} \\ C_2 = \beta + \frac{2a\beta - 2b\tau + 4\tau^2 + 2\tau\sqrt{b^2 - 4b\tau + 4a\beta + 4\tau^2}}{2a} & \textit{obtenue avec la } 2^{i\grave{e}me} \textit{ contrainte.} \end{array} \right.$$

La figure 2 illustre l'évolution des tailles de paquets, pour un macro-pipeline de taille 1024, en fonction de la taille critique imposée Λ.

3. Validations expérimentales

L'implémentation utilise la bibliothèque OPIUM [2] [3], dans laquelle a été intégrée une primitive calculant la suite de taille de paquets pour la factorisation de Cholesky, et la bibliothèque LOCCS [3] [4], modifiée pour prendre en compte un découpage non constant du macro-pipeline. Les mesures de performances sont réalisées sur un calculateur SP2 IBM avec les paramètres : $\beta = 1.4\ 10^{-4}$, $\tau = 7.3\ 10^{-7}$, $\tau_\times = \tau_\div = 3.8\ 10^{-7}$. Le calcul est traité en *double précision* sur 8 noeuds fins. On se restreint dans cette étude expérimentale à l'utilisation du noyau BLAS 1.

Les figures 3 et 4 (sans BLAS 1) montrent que l'influence du recouvrement devient significative dès que la masse de calcul à traiter par processeur est suffisante (pour de petits macro-pipelines, le nombre de paquets générés est trop faible pour avoir un recouvrement efficace).

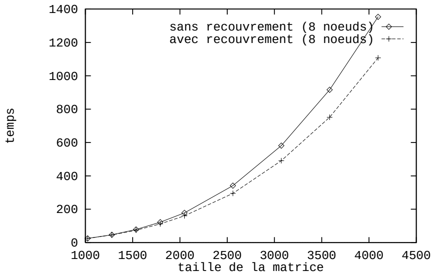

Figure 3. Recouvrement sans BLAS 1

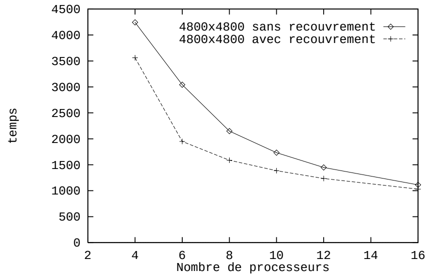

Figure 4. Extensibilité sans BLAS 1

[2] Optimal Packet sIze compUtation Methods.

[3] Low Overhead Communication and Computation Subroutines.

Les tests menées avec les optimisations BLAS 1 conduisent à des résultats similaires, mais l'efficacité du recouvrement est mise en évidence pour des matrices plus grandes (figures 5 et 6). Expérimentalement on retrouve la taille critique attendue pour un BLAS 1 (de l'ordre de 150 éléments); on atteint 64 Mflops sur 4 noeuds et 43 Mflops sur 12 noeuds.

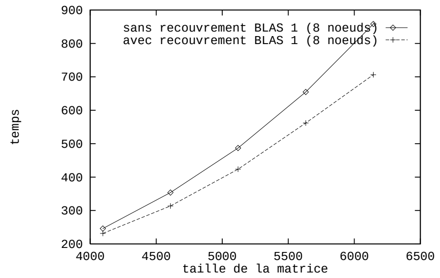

Figure 5. Recouvrement avec BLAS 1

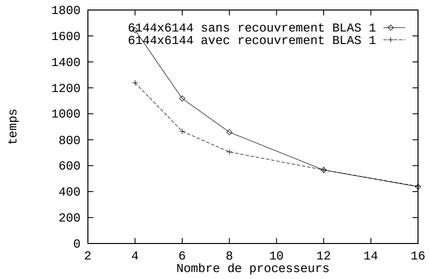

Figure 6. Extensibilité avec BLAS 1

4. Conclusion

Dans cet article, nous avons présenté et validé expérimentalement une méthode pour recouvrir les communications dans le cas de l'exemple de la factorisation de Cholesky. Les résultats obtenus montrent l'efficacité de la méthode. La prise en compte des optimisations de type BLAS doit être poursuivie pour les BLAS de niveau 2 et 3, l'objectif est ici d'intégrer ces techniques de recouvrement dans des algorithmes par blocs comme ceux proposés dans ScaLAPACK [2], et de comparer les 2 versions.

La suite de ce travail consiste tout d'abord à généraliser ces résultats au cas de la factorisation de Cholesky pour des matrices creuses quelconques; dans ce contexte plus général d'irrégularité, l'objectif sera de calculer, à l'aide d'un *schéma adaptatif*, la suite optimale de taille de paquets. On intégrera ensuite ces techniques dans un schéma de compilation automatique pour le langage HPF (plateforme Adaptor). L'intégration des LOCCS a déjà été réalisée [1].

Bibliographie

1. T. Brandes and F. Desprez. Implementing Pipelined Computation and Communication in an HPF Compiler. In *Proceedings of Euro-Par'96*, number 1123 in LNCS, pages 459–462. Springer Verlag, 1996.
2. J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. LAPACK Working Note: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performances. Technical Report 95, Department of Computer Science - University of Tennessee, 1995.
3. F. Desprez, P. Ramet, and J. Roman. Optimal grain size computation for pipelined algorithms. In *Proceedings of Euro-Par'96*, number 1123 in LNCS, pages 165–172. Springer Verlag, 1996.
4. F. Desprez and B. Tourancheau. LOCCS: Low Overhead Communication and Computation Subroutines. *Future Generation Computer Systems*, 10(2):279–284, June 1994.
5. L.Colombet, P.Michallon, and D.Trystram. Parallel matrix-vector product on rings with a minimum of communications. In *Parallel Computing*, number 22 in Elsevier Science, pages 289–310, 1996.
6. H. Ohta, Y. Saito, M. Kainaga, and H. Ono. Optimal Tile Size Adjustement in Compiling General DOACROSS Loop Nests. In ACM Press, editor, *International Conference on Supercomputing*, pages 270–279, Barcelona, Spain, July 1995. ACM SIGARCH.
7. D. Park. Adaptive granularity: Transparent integration of fine-grain and coarse grain communications. Technical report, Computer Science Departement, University of Southern California, 1996.
8. D. W. Walker. Portable programming within a message-passing model: the fft as an example. In *The Third Conference On Hypercube Concurrent Computers and Applications*, volume II - Applications. Geoffrey Fox California Institute of Technology, 1988.