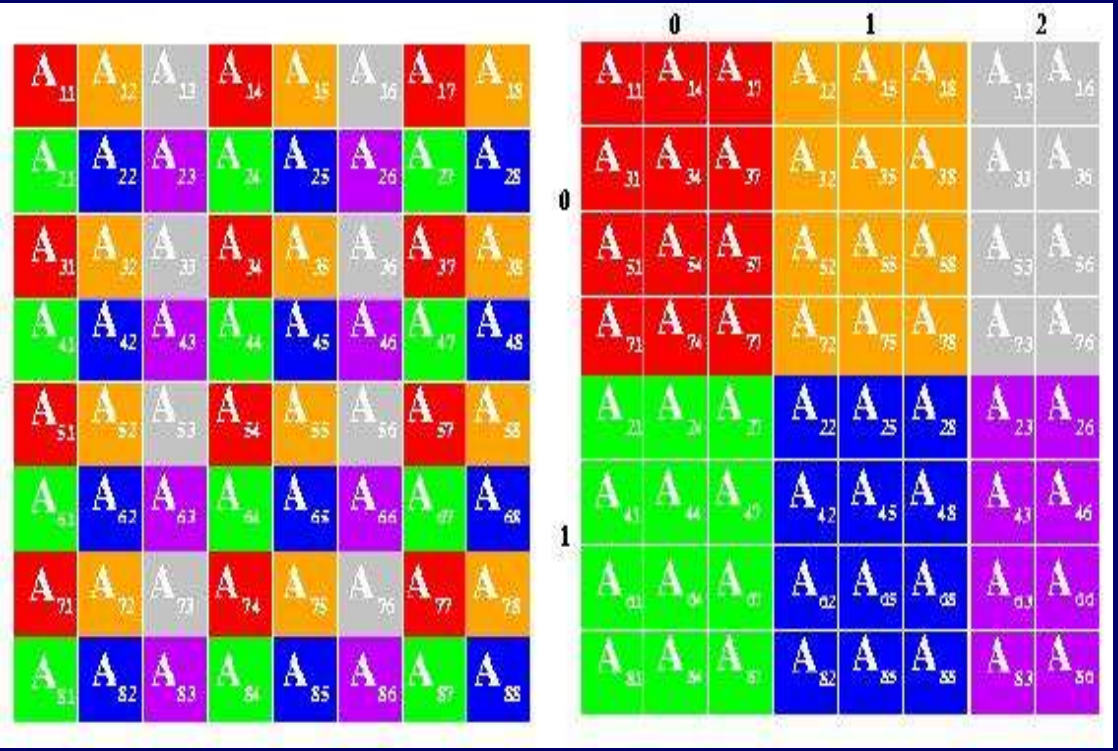
Cours 4. Analyse d'algorithmes avec prise en compte des communications

- Analyse de schémas de communication globale
- Exemple du produit de matrices
- FFT

www-id.imag.fr/Laboratoire/Membres/Roch_Jean-Louis/perso.html/COURS/

Distribution bloc-cyclique bi-dimensionnelle

Sur 6 Processeurs, agencés en grille 2x3



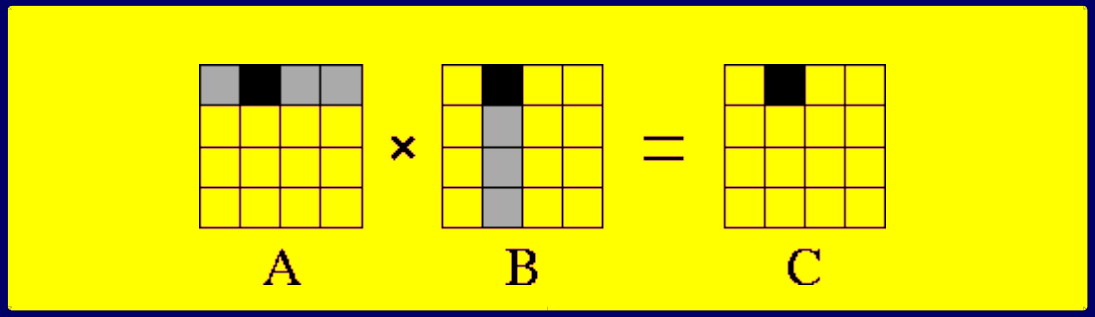
Exemple de synthèse : produit de matrices

I. Distribution de données: ligne-colonne

II. Distribution de données: bloc bi-dimensionnel

III. Distribution de calculs

Algorithme 1: ligne-colonne



- A(i, _) : découpée en K blocs de n/K lignes:
- B(_, j) : découpée en K blocs de n/K colonnes
  - Répliquée sur tous les processeurs
- C(i, _) : découpée en K blocs de n/K lignes

- Affectation des calculs sur C(i,_) au même proc
- Coût : $T_\infty = (n/K).n^2 = n^3/K$ $T_1 = = n^3$ opérations

Algorithme ligne-colonne
Circulation des colonnes de B

- Algorithme en K étapes : sur le proc i (0<=i<K<=n)
  - Init : C(i, _) = 0 ;
  - For (k=0; k<K; k++) {

    w C(i, i+k % K) = A(i, _)*B(_, i+k % K) ;

  - }

Algorithme ligne-colonne
Circulation des colonnes de B

- Algorithme en K étapes : sur le proc i (0<=i<K<=n)
  - Init : C(i, _) = 0 ;
  - For (k=0; k<K; k++) {

    w C(i, i+k % K) = A(i, _)*B(_, i+k % K) ;

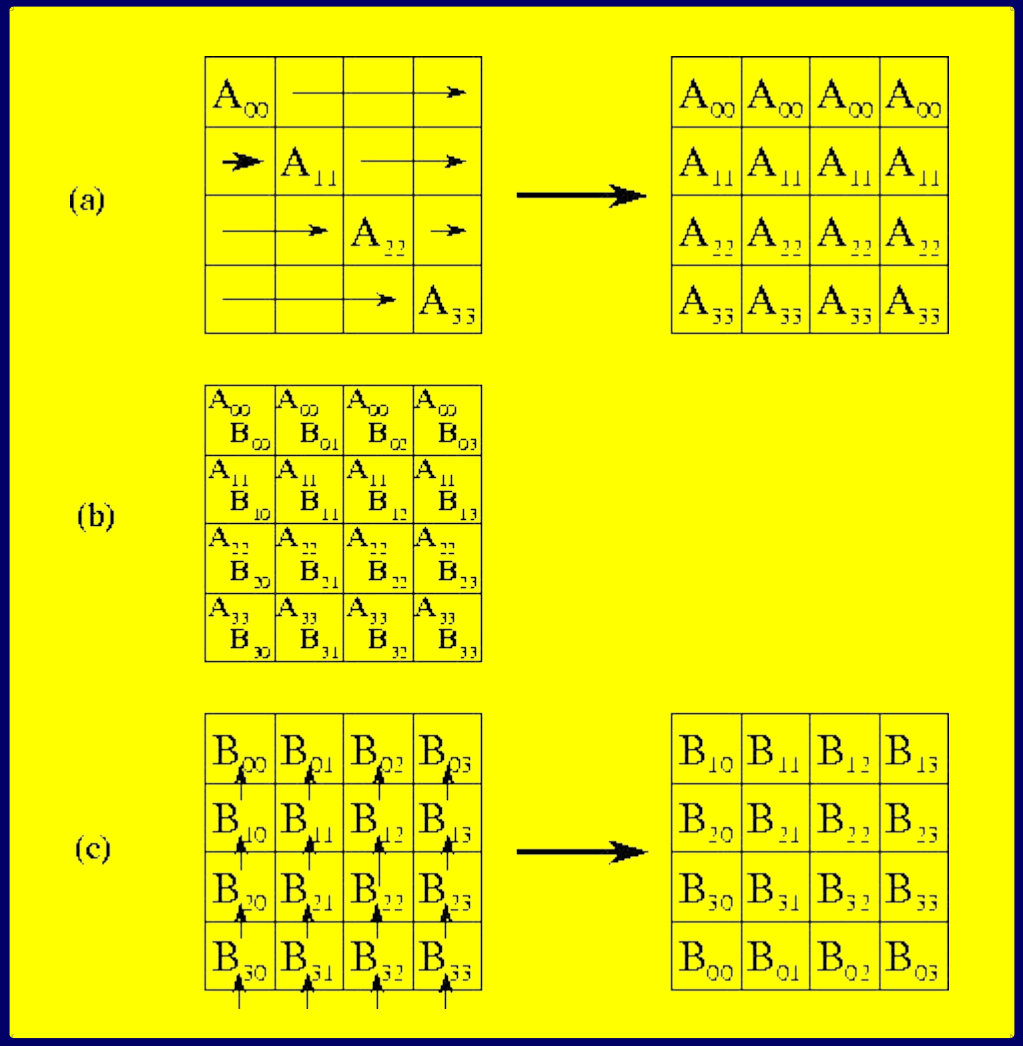
  - }
- Coût : $T_\infty = K.(n/K)^2.n = n^3/K$ et $T_1 = n^3$
- Communications: $C_\infty = n^2/K + K.n^2/K = n^2$ **et $C_1 = n^2.K$**
- **Avec K=p <= n processeurs** : $T_\infty = n^3/p$ et **$C_1 = n^2.p$**

Algorithme ligne-colonne
Circulation des colonnes de B

- Algorithme en K étapes : sur le proc i (0<=i<K<=n)
  - Init : C(i, _) = 0 ; localB=B(_,i); tmpB = buffer(**n^2/K**);
  - For (k=0; k<K; k++) {
    - async_send (localB, i-1%K) ; async_recv(tmpB, i+1%K) ;
    - C(i, i+k % K) = A(i, _)*localB ;
    - Sync ; swap (localB, tmpB);
  - }
- Coût : $T_\infty = K.(n/K)^2.n = n^3/K$ et $T_1 = n^3$
- Communications: $C_\infty = n^2/K + K.n^2/K = n^2$ **et $C_1 = n^2.K$**
- **Avec K=p <= n processeurs** : $T_\infty = n^3/p$ et **$C_1 = n^2.p$**

Algorithme par blocs bidimensionnel

A : circulation en ligne



B : circulation en colonne

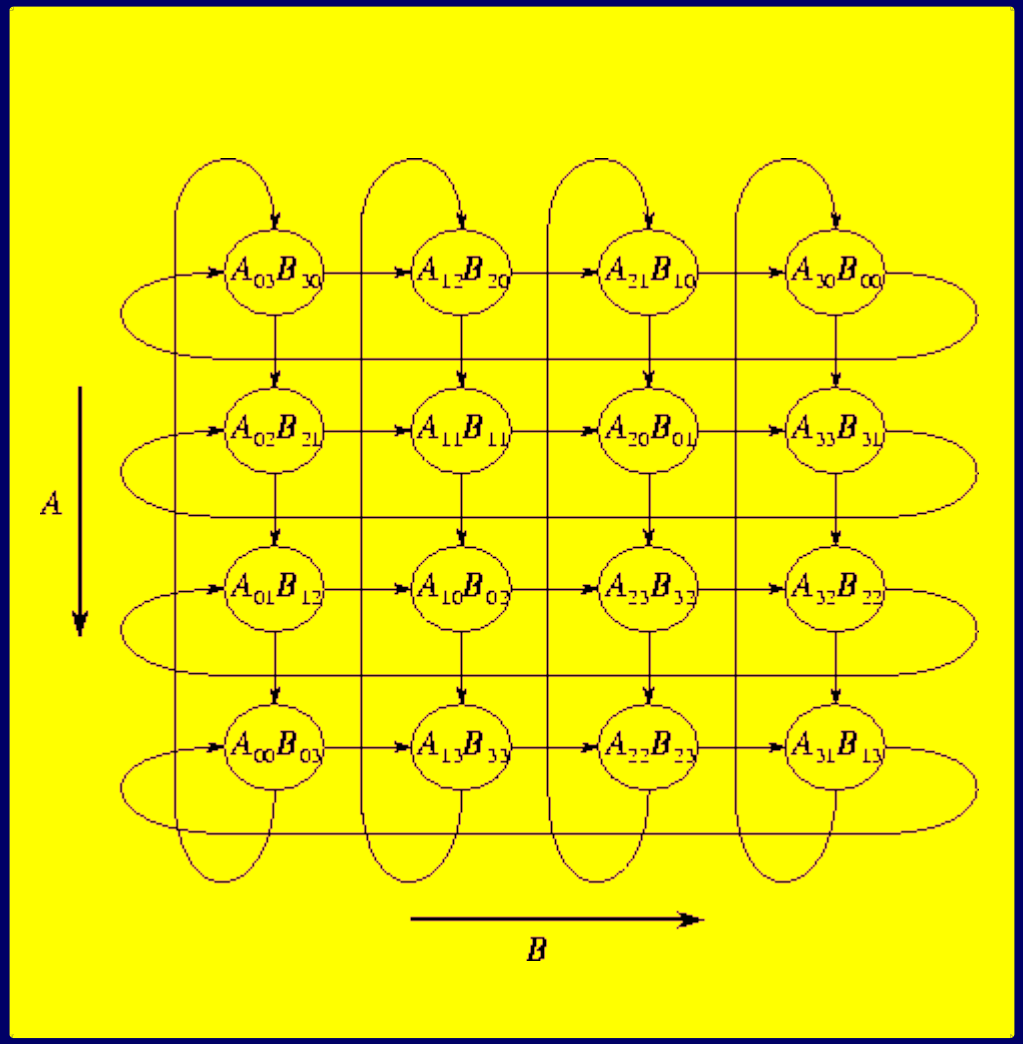
KxK blocs de taille (n/K)x(n/K)

$0<K<=n$

- $K^2=p$
- $K=\sqrt{p}$

Produit de matrices en bloc : KxK blocs

A : circulation
en ligne



B : circulation
en colonne

Algorithme par bloc

- - Calcul C(i,j) = Σ A(i,k)*B(k,j) sur un même proc
- - Pipeline: circulation A par ligne; circulation B par colonne
- - Init: Attention : on démarre avec A(i,i) sur chaque proc.
- Processeur (i,j) :
  - § Init : wait localA= A(i,i); localB=B(i,j); tmpA= ...; tmpB= ...
  - § For (k=1; k<K; k++) {
    - w async_send (localA, P(i,j-1%K)) ; async_recv(tmpA, P(i,j+1%K) ;
    - w async_send (localB, P(i-1%K,j)) ; async_recv(tmpB, P(i+1%K, j) ;
    - w C(i, j) += localA*localB ;
    - w Sync ; swap (localB, tmpB); swap (localA, tmpA); }

Algorithme par bloc

- - Calcul C(i,j) = Σ A(i,k)*B(k,j) sur un même proc
- - Pipeline: circulation A par ligne; circulation B par colonne
- - Init: Attention : on démarre avec A(i,i) sur chaque proc.
- Processeur (i,j) :
  - § Init : wait localA= A(i,i); localB=B(i,j); tmpA= ...; tmpB= ...
  - § For (k=1; k<K; k++) {
    - w async_send (localA, P(i,j-1%K)) ; async_recv(tmpA, P(i,j+1%K) ;
    - w async_send (localB, P(i-1%K,j)) ; async_recv(tmpB, P(i+1%K, j) ;
    - w C(i, j) += localA*localB ;
    - w Sync ; swap (localB, tmpB); swap (localA, tmpA); }
- Coût : $T_\infty = K.(n/K)^3 = n^3/K^2$ et $T_1 = n^3$
- Communications: $C_\infty = K.n^2/K^2 = n^2/K$ **et $C_1 = n^2.K$**
- **Avec $K=\sqrt{p}$ ($p <= n^2$)processeurs** : $T_\infty = n^3/p$ et **$C_1 = n^2.\sqrt{p}$**

Algorithme par distribution des calculs

- - A et B découpées en KxK blocs bidimensionnels
- - => K^3 calculs parallèles : tmp(i, k, j) = A(i,k) * B(k,j)

- - Puis réduction : C(i, j) = $\Sigma_{\kappa=0..K-1}$ tmp(i,k,j) (en arbre)

Algorithme par blocs et distribution des calculs

- A et B découpées en KxK blocs bidimensionnels
- => K^3 calculs parallèles : tmp(i, k, j) = A(i,k) * B(k,j)

  Coût : $T_\infty = (n/K)^3 = n^3/K^3$ et $T_1 = n^3$ avec K^3 procs

  Comm.: $C_\infty = n^2/K^2$ et $C_1 = n^2.K$
- Puis réduction : $C(i, j) = \sum_{\kappa=0..K-1} tmp(i,k,j)$ (arbre)

  Coût : $T_\infty = (n/K)^2 .\log K$ et $T_1 = K^2.K.(n/K)^2 = n^2.K$

  Comm.: $C_\infty = n^2/K^2 .\log K$ et $C_1 = n^2.K$

- Avec $K=p^{1/3}$ ($p <= n^3$)processeurs : $T_\infty = n^3/p$ et $C_1 = n^2.p^{1/3}$

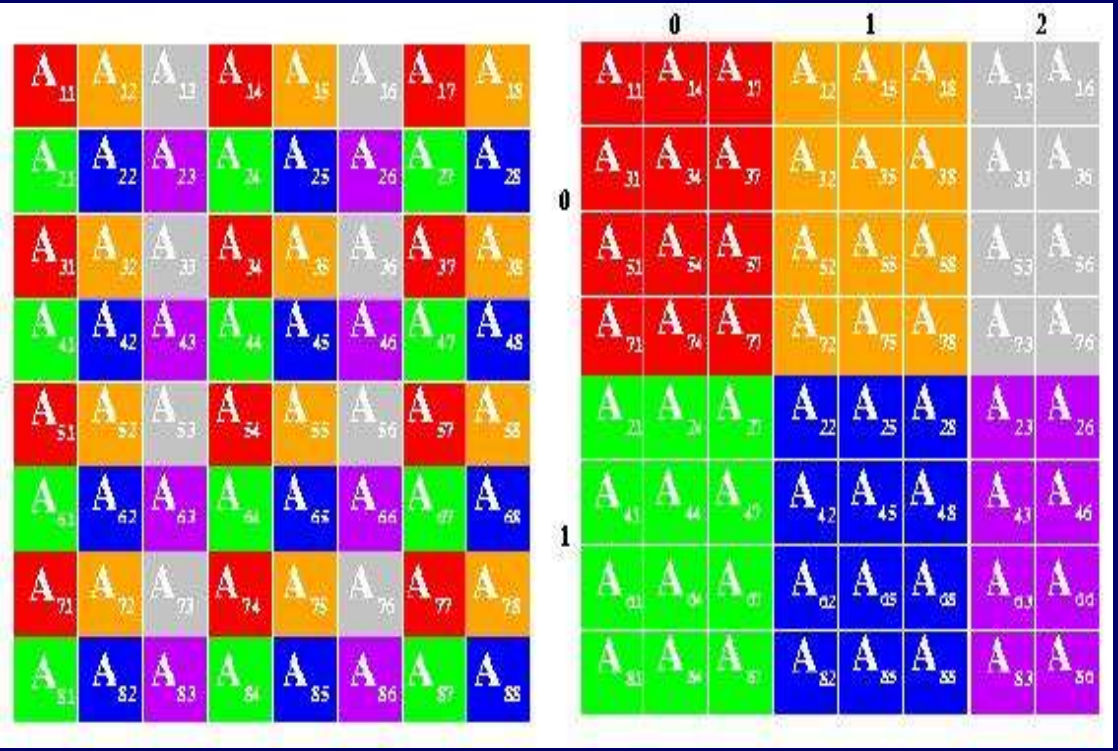
Algorithme par bloc et distribution des calculs

- - Pipeline des accès pour masquer le facteur log K
- - On choisit K >> $p^{1/3}$ (i.e. Parallelisation plus fine)
- - Chaqueprocesseur peut ainsi anticiper les sommations

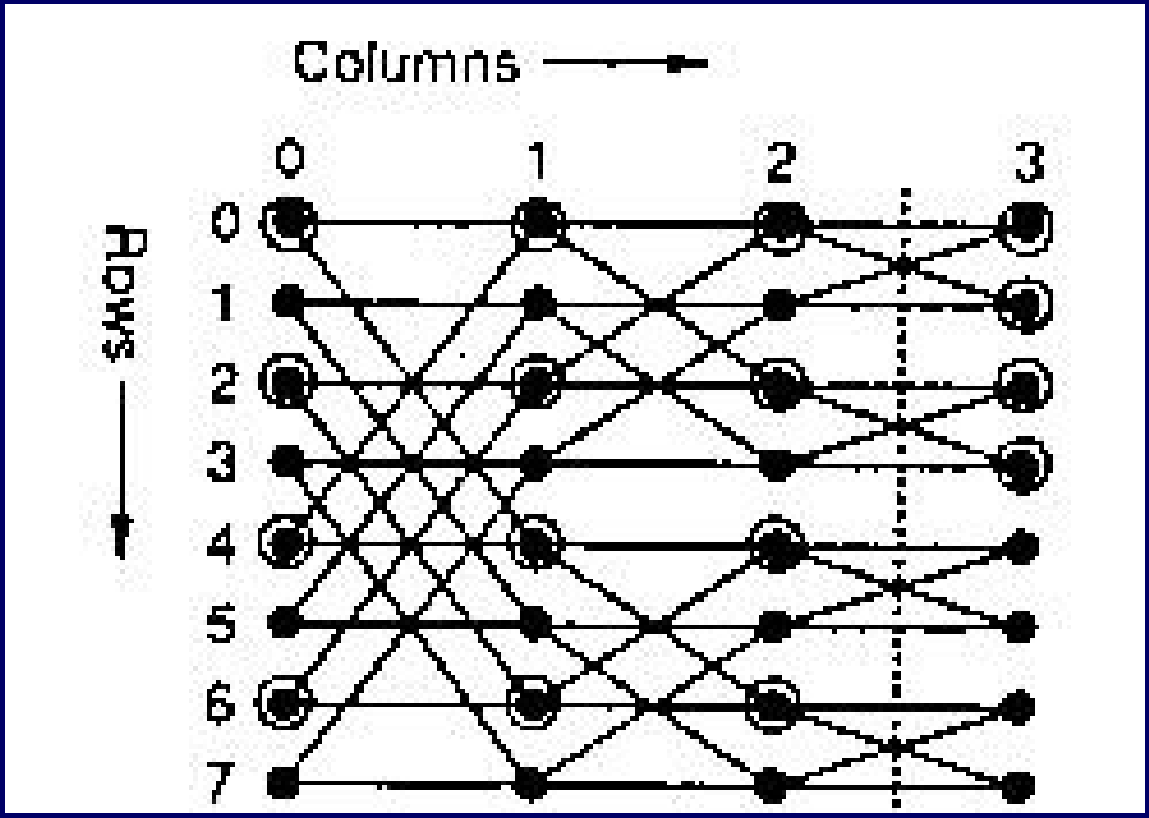
- => Avec $p=n^3$ processeurs : $C_\infty = n^2/K^2$ et $C_1 = n^2.K$ soit $C_p=n^{4/3}$ au max. par proc.
- => Distribution bloc-cyclique bi-dimensionnel

Distribution bloc-cyclique bi-dimensionnelle

Sur 6 Processeurs, agencés en grille 2x3



Exemple 3 : FFT



FFT : performances

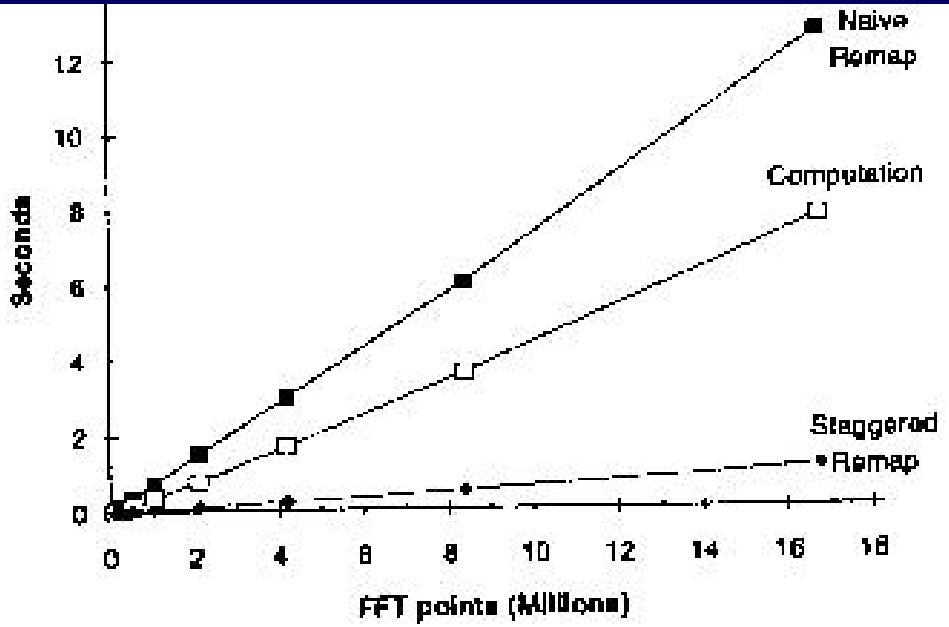


Figure 6: *Execution times for FFTs of various sizes on a 128 processor CM-5. The compute curve represents the time spent computing locally. The bad remap curve shows the time spent remapping the data from a cyclic layout to a blocked layout if a naive communication schedule is used. The good remap curve shows the time for the same remapping, but with a contention free communication schedule, which is an order of magnitude faster. The X axis scale refers to the entire FFT size.*

« LogP : Towards a realistic model of Parallel Computation » D. Culler & al.

FFT : redistribution

$T_1 = \sqrt{n}$

$T_\infty = \sqrt{n}.\log n$

$C_1 = n$