

High Performance Linpack (HPL)

LIBERATE IT

HPL – « High Performance Linpack »

- Source : <http://www.netlib.org/benchmark/hpl>
- Abusivement appeler linpack
- Le benchmark absolu ?
- Benchmark de référence servant à l'établissement du classement du Top500 (www.top500.org)
- Résout un système dense d'équation linéaire $Ax+B$ de dimension $N \times N$ sur une architecture à mémoire distribuée, en double précision.
- Le résultat brut est un temps de restitution et une performance pic (quantité absolue)
 - Elle est à ramener à la performance max. du système pour en déduire une efficacité (quantité relative)
 - Ce qui n'est pas fait dans le Top500 ...

- LINear algebra PACKage (1974)

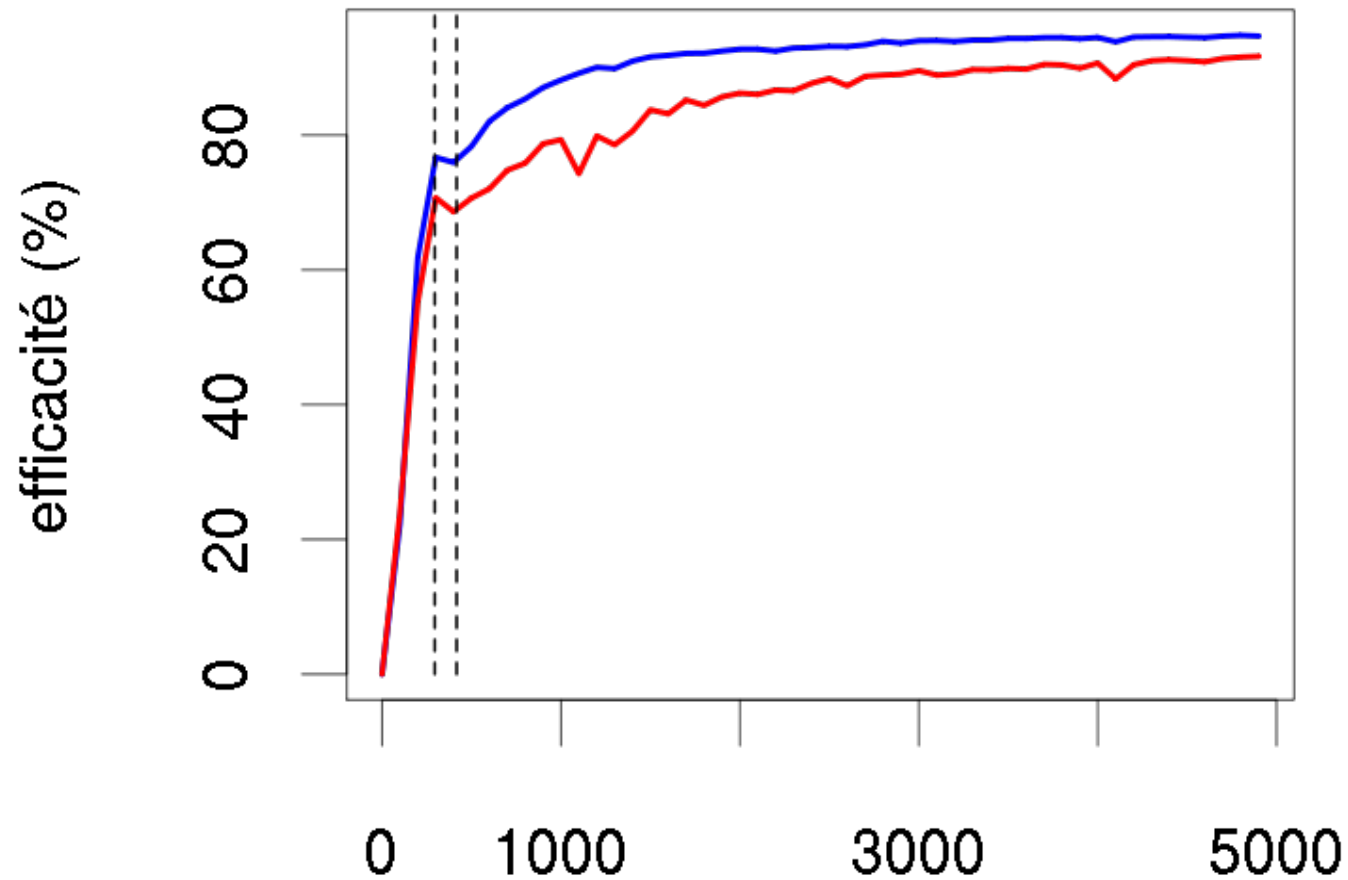
- Fortran66
- Historiquement Linpack une bibliothèque mathématiques pour la résolution de problèmes en algèbre linéaire.
- Encore un benchmark par accident !
 - Annexe B du manuel de l'utilisateur : permettre aux utilisateurs d'estimer les temps de calculs de résolution de leur problème !
- 2 benchmarks : linpack-100 (1977) , linpack-1000 (1986)
- Vient alors HPL

UNIT = 10**6 TIME/(1/3 100**3 + 100**2)						
Facility	TIME N=100 secs.	UNIT micro- secs.	Computer	Type	Compiler	
NCAR	14.0	.049	0.14	CRAY-1	S	CFT, Assembly BLAS
LASL	4.67	.148	0.43	CDC 7600	S	FLN, Assembly BLAS
NCAR	3.57	.192	0.56	CRAY-1	S	CFT
LASL	3.27	.210	0.61	CDC 7600	S	FTN
Argonne	2.31	.297	0.86	IBM 370/195	D	H
NCAR	1.81	.359	1.05	CDC 7600	S	Local
Argonne	1.77	.388	1.33	IBM 3033	D	H
NASA Langley	1.40	.489	1.42	CDC Cyber 175	S	FTN
U. Ill. Urbana	1.56	.506	1.47	CDC Cyber 175	S	Ext. 4.6
LLL	1.24	.554	1.61	CDC 7600	S	CHAT, No optimize
SLAC	1.19	.579	1.69	IBM 370/168	D	H Ext., Fast mult.
Michigan	1.09	.631	1.84	Amdehl 470/V6	D	H
Toronto	.772	.690	2.59	IBM 370/165	D	H Ext., Fast mult.
Northwestern	.477	1.44	4.20	CDC 6600	S	FTN
Texas	.561	1.93*	5.63	CDC 6600	S	RUN
China Lake	.521	1.95*	5.69	Univac 1110	S	V
Yale	.265	2.59	7.53	DEC KL-20	S	F20
Bell Labx	.077	3.46	10.1	Honeywell 6080	S	Y
Wisconsin	.107	3.49	10.1	Univac 1110	S	V
Iowa State	.194	3.54	10.2	Itel AS/5 mod3	D	H
U. Ill. Chicago	.064	4.10	11.9	IBM 370/158	D	G1
Purdue	.045	5.69	16.6	CDC 6500	S	FUN
U. C. San Diego	.040	13.1	38.2	Burroughs 6700	S	H
Yale	.040	17.1*	49.9	DEC KA-10	S	F40

Performances d'un HPL

- De quoi dépendent les performances d'un HPL :
 - L'essentiel des calculs est une DGEMM (BLAS-3). De l'efficacité de la DGEMM dépend l'efficacité du Linpack
 - L'efficacité des DGEMM aujourd'hui sur des Nehalem dépasse les 90%
 - Les données sont spatialement et temporellement locales
 - C'est indirectement un bench de calcul (CPU), à condition d'une DGEMM optimale ; c'est un benchmark de DGEMM !!!
 - La taille du problème N :
 - L'efficacité d'une DGEMM (et celle comme HPL) dépendent de la taille du problème : plus il est grand, plus efficace est le calcul.
 - Plus dense est la matrice, mieux on peut recouvrir les communications par du calcul.
 - N est fonction du nombre de cœurs.
 - De l'interconnect.

Performance d'une DGEMM avec N



Exemple : clusters de nehalem en juin 2009

	<i>Site</i>		<i>Cores</i>	<i>RMax</i>	<i>RPeak</i>	<i>Eff.</i>	<i>Interconnect Family</i>
10	Forschungszentrum Juelich (FZJ)	Bull SA	26304	274800	308283	89%	Infiniband QDR
16	SciNet/University of Toronto	IBM	30240	168600	306029	55%	Gigabit Ethernet
38	Commissariat a l'Energie Atomique (CEA)/CCRT	Bull SA	8576	91190	100510	90%	Infiniband DDR

Côté « pratique » ...

– Portage

- Implémentation en C
- Relativement facile
 - Makefile a adapté selon l'architecture
 - Nécessite un bibliothèque de BLAS (netlib, GotoBLAS, Intel MKL,...) ainsi qu'une implémentation MPI-1

– Execution

- Auto-vérifiant
- Nécessite un fichier d'entrée : HPL.dat
 - 17 paramètres importants
 - Taille du système, des blocs, décomposition, « pattern » de communication
 - Un bon point de départ de partir de fichiers publiés pour des architecture comparable sur le site de HPCC dont il fait parti (<http://icl.cs.utk.edu/hpcc/index.html>)
 - Plusieurs étapes dans l'exécution d'un HPL
 - Phase de préparation (intégrité des noeuds)
 - Exécution d'un run moyen
 - Exécution d'un run long (quelques %)
 - Optimisation du fichier d'entrée, la chasse < 1%

HPL.dat : un exemple (canonique)

HPLinpack benchmark input file
 Innovative Computing Laboratory, University of Tennessee
 HPL.out output file name (if any)
 6 device out (6=stdout,7=stderr,file)
 4 # of problems sizes (N)
 29 30 34 35 Ns
 4 # of NBs
 1 2 3 4 NBs
 0 PMAP process mapping (0=Row-,1=Column-major)
 3 # of process grids (P x Q)
 2 1 4 Ps
 2 4 1 Qs
 16.0 threshold
 3 # of panel fact
 0 1 2 PFACTs (0=left, 1=Crout, 2=Right)
 2 # of recursive stopping criterium
 2 4 NBMINS (>= 1)
 1 # of panels in recursion
 2 NDIVs
 3 # of recursive panel fact.
 0 1 2 RFACTs (0=left, 1=Crout, 2=Right)
 1 # of broadcast
 0 BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
 1 # of lookahead depth
 0 DEPTHs (>=0)
 2 SWAP (0=bin-exch,1=long,2=mix)
 64 swapping threshold
 0 L1 in (0=transposed,1=no-transposed) form
 0 U in (0=transposed,1=no-transposed) form
 1 Equilibration (0=no,1=yes)

Détermination de N et $p \times q$

- $p \times q$: décomposition

- ' $p \times q$ ' est forcément égal au nombre de processeurs (coeurs) utilisés avec
 - $p < q$
 - p et q proche

- N : dimension du système

- Aussi grand que possible ; mais il faut tenir compte de l'empreinte mémoire MPI (peut-être importante selon les souches).
 - Si m est la mémoire disponible par coeur en MiB , n le nombre de coeurs (donc $p \times q = n$) et e la fraction de mémoire que l'on souhaite utiliser. Alors :

$$N = 1024 \times \text{racine} (e \cdot p \cdot q / 8)$$

- Le nombre d'opérations est connu : $F_{\text{ops}} = \frac{2}{3} \cdot N^3 + 2 \cdot N^2$ FLOPS
- Connaissant l'efficacité d'une dgemm ε , on peut déduire le temps (en négligeant les communications !) tout au moins le borner.

Estimations ...

- Performance max du système : $P_{\text{peak}} = p \times q \times (\text{inst/cycle}) \times \text{Freq}$
- Perf max. théorique du système : $P_{\text{max}} < \varepsilon P_{\text{peak}}$ secondes
- $T = F_{\text{ops}} / P_{\text{max}}$ secondes
- Ex. Titane (CCRT) – Nehalem 2.93 GHz
 - 1072 noeuds, 24 GiB par noeuds ; perf. Max. théorique : 100.5 TFLOPS/sec
 - Mémoire totale disponible : 25728 GiB
 - Pour 72.7% de mémoire : $N=1024 \text{ racine}(0.727 \times 1072 \times 24 \times 1024)=1584438$
 - Pour efficacité de 85% : 30911 secondes soit 8 h 35 min.
 - En réalité : 91.19 TFLOPS/sec soit 90.73% d'efficacité (avec turbomode).

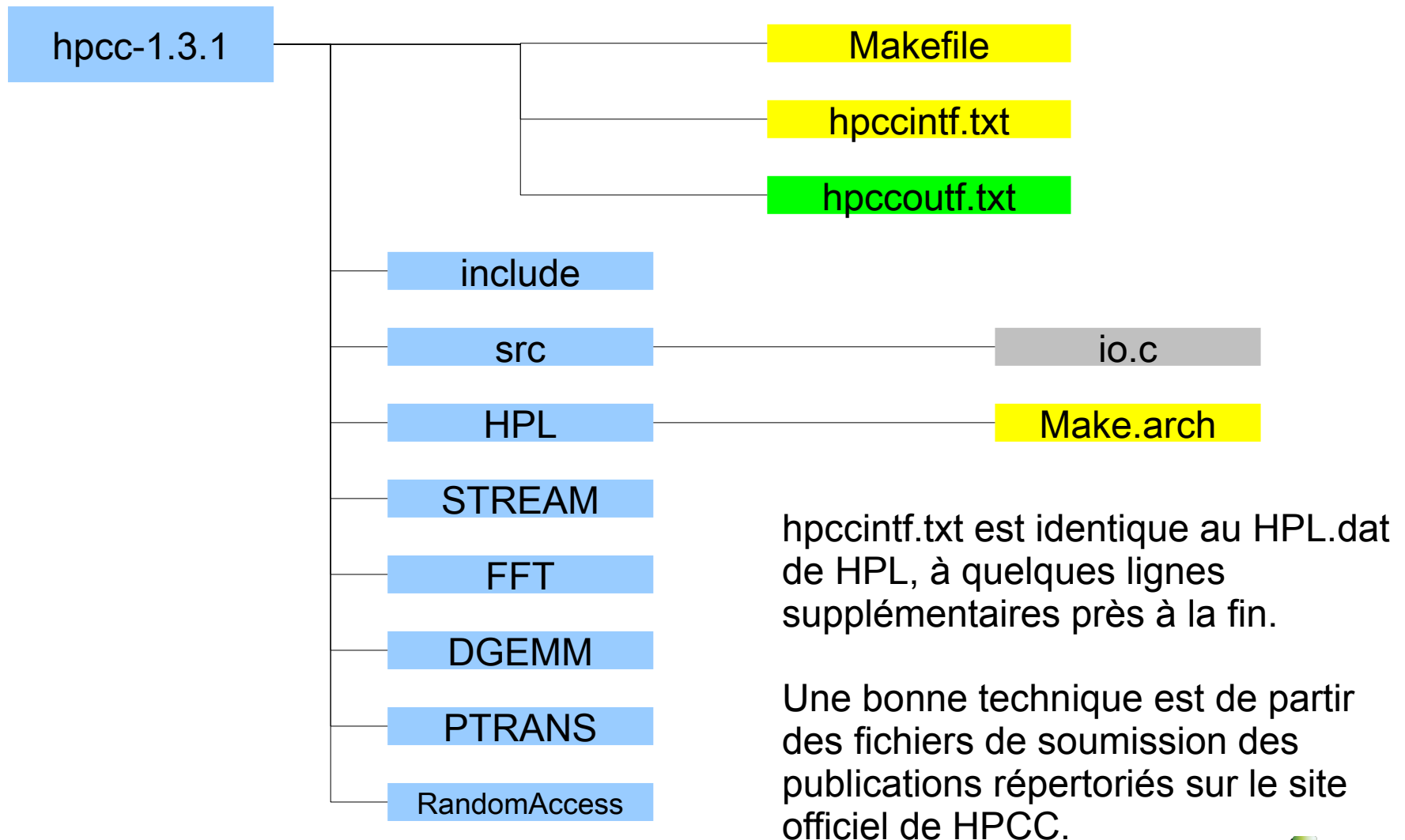
```
=====
T/V      N  NB  P  Q      Time      Gflops
-----
WR01L2L4 1585024 128 67 128      29112.29      9.119e+04
-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=      0.0003187 ..... PASSED
=====
```

HPC Challenge (HPCC)

LIBERATE IT

- <http://icl.cs.utk.edu/hpcc/>
- HPC Challenge
- Suite de microbenchmarks/mid-level range
 - HPL, DGEMM, Stream, PTRANS, RandomAccess, FFT, Beff
- Motivation
 - Pour étudier les performances des architectures HPV en utilisant des « kernels » plus stressants du point de vue des « pattern » d'accès mémoire que peut l'être HPL.
 - Candidat au remplacement de HPL
 - Au CPU ajouter le stress mémoire et de l'interconnect
 - Vers le pétaflopique !

- Portage :
 - Écrit en C
 - Relativement simple ; ce base sur HPL
 - Mêmes dépendances donc : une bibliothèque de BLAS, une implémentation MPI. Non obligatoire : une implémentation de fftw (fftw.org ou Intel MKL)
- 2 types de runs
 - Pour des runs publiables :
 - « Baseline » (ou « as-is »)
 - « Optimized »
 - Sous certaines conditions
 - 128 résultats par runs (là où HPL en fourni un seul !)
 - Les résultats sont auto-vérifiants

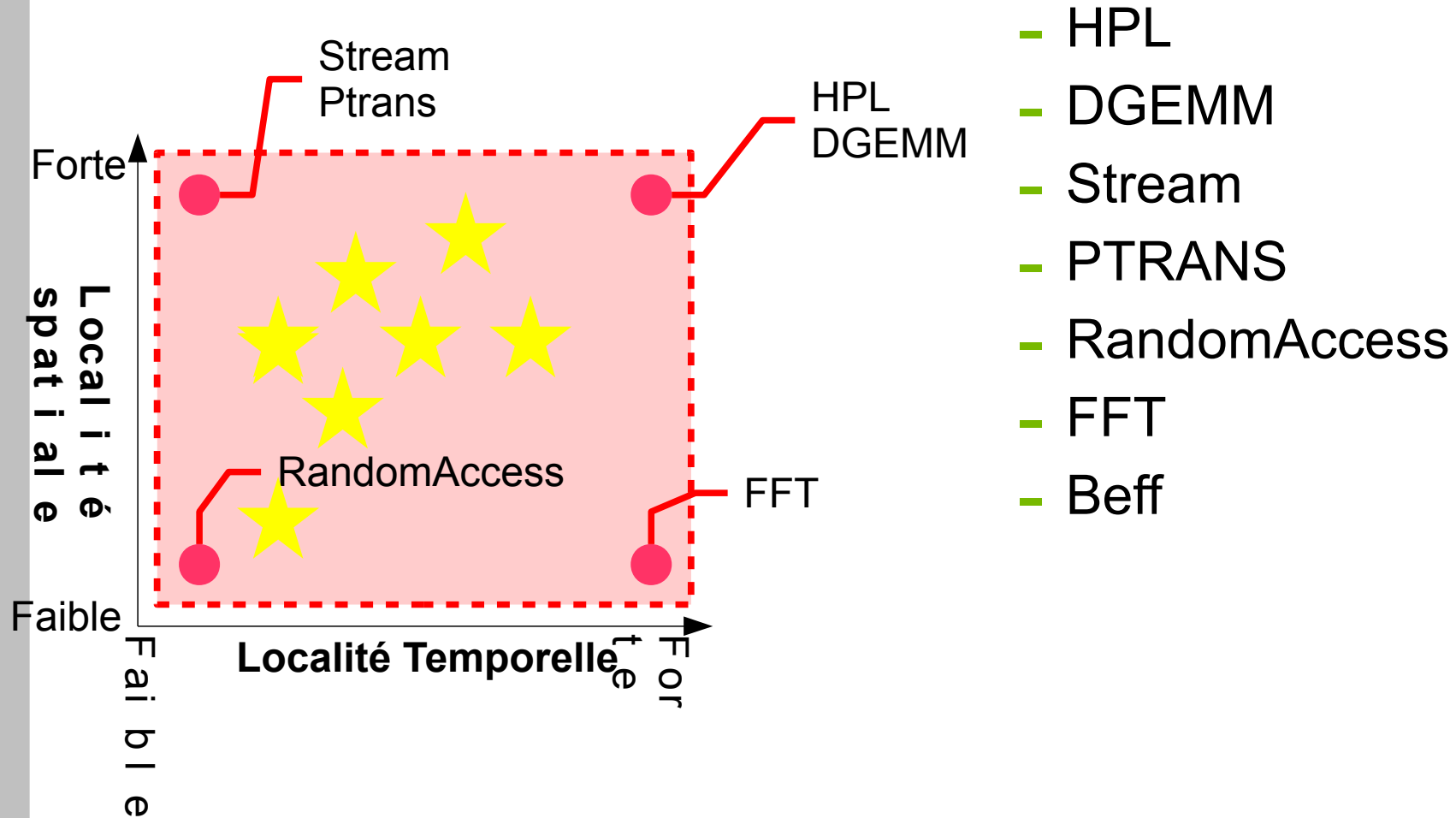


- Certains demandent déjà (incorrigibles..) de réduire HPCC à un seul score et faisant une moyenne pondérée des 7 scores principaux :

$$\text{Score} = W1 * \text{LINPACK} + W2 * \text{DGEMM} + W3 * \text{FFT} + W4 * \text{Stream} + W5 * \text{RA} + W6 * \text{Ptrans} + W7 * \text{BW/Lat}$$

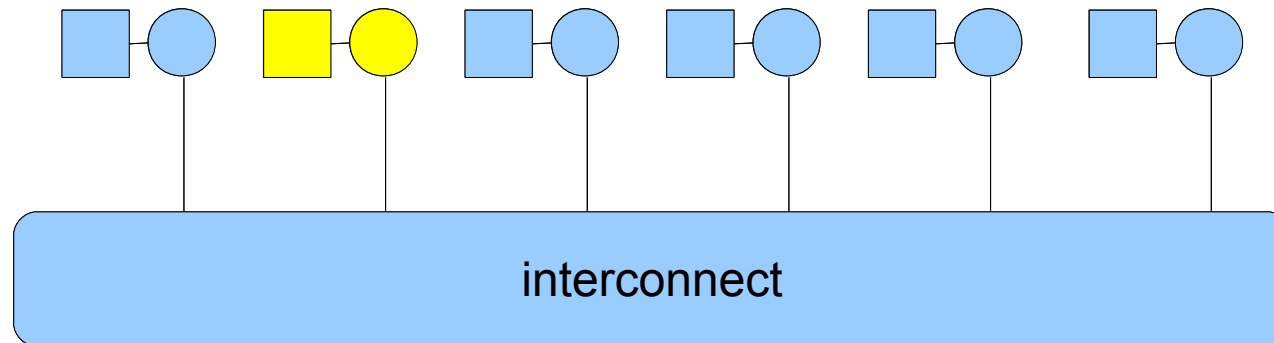
- Quel choix pour les poids W_i ?
 - Ce qui se cache derrière c'est bien sûr le « fantôme du benchmarkeur »
 - A chaque type d'application son ensemble de $\{W_i\}$...

Suite de tests



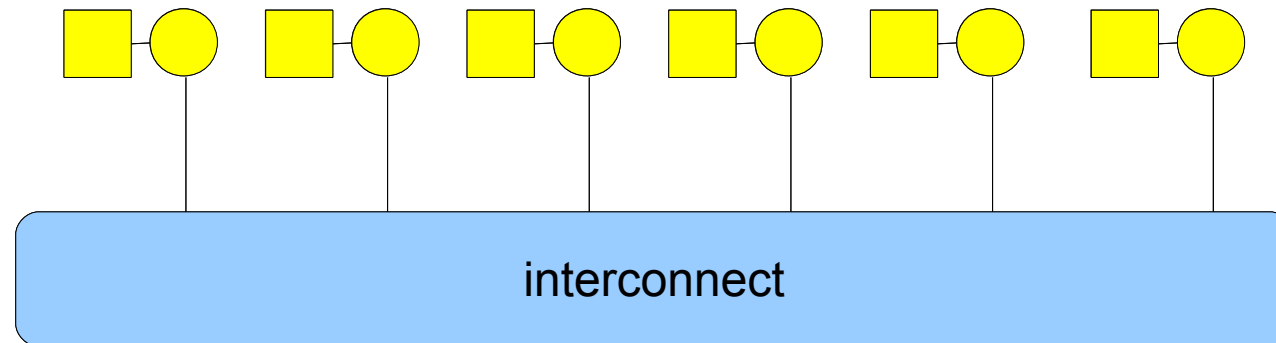
Type de runs

- Chaque code (s'il le permet) effectue 3 tests différents :
 - « single »
 - 1 seul processus
 - « embarrassingly parallel »
 - Tous les processus indépendamment
 - « global »
 - Tous les processus partagent la charge



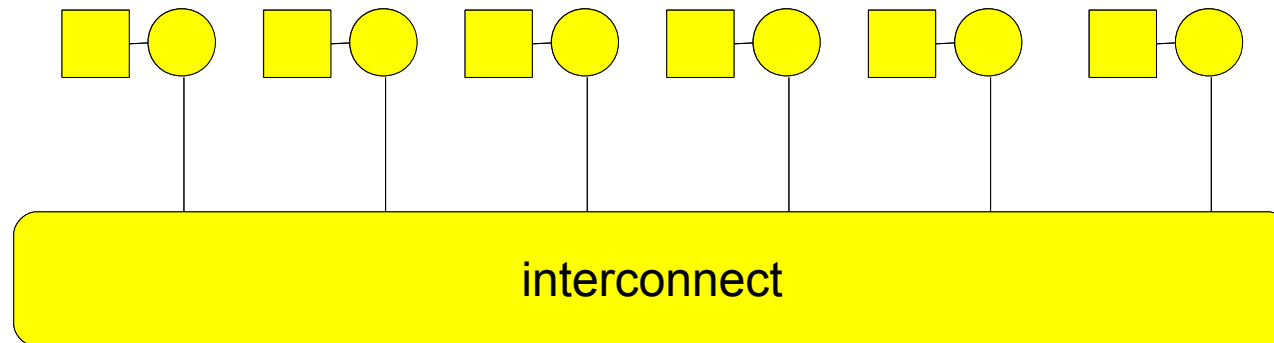
Type de runs

- Chaque code (s'il le permet) effectue 3 tests différents :
 - « single »
 - 1 seul processus
 - « embarrassingly parallel »
 - Tous les processus indépendamment
 - « global »
 - Tous les processus partagent la charge

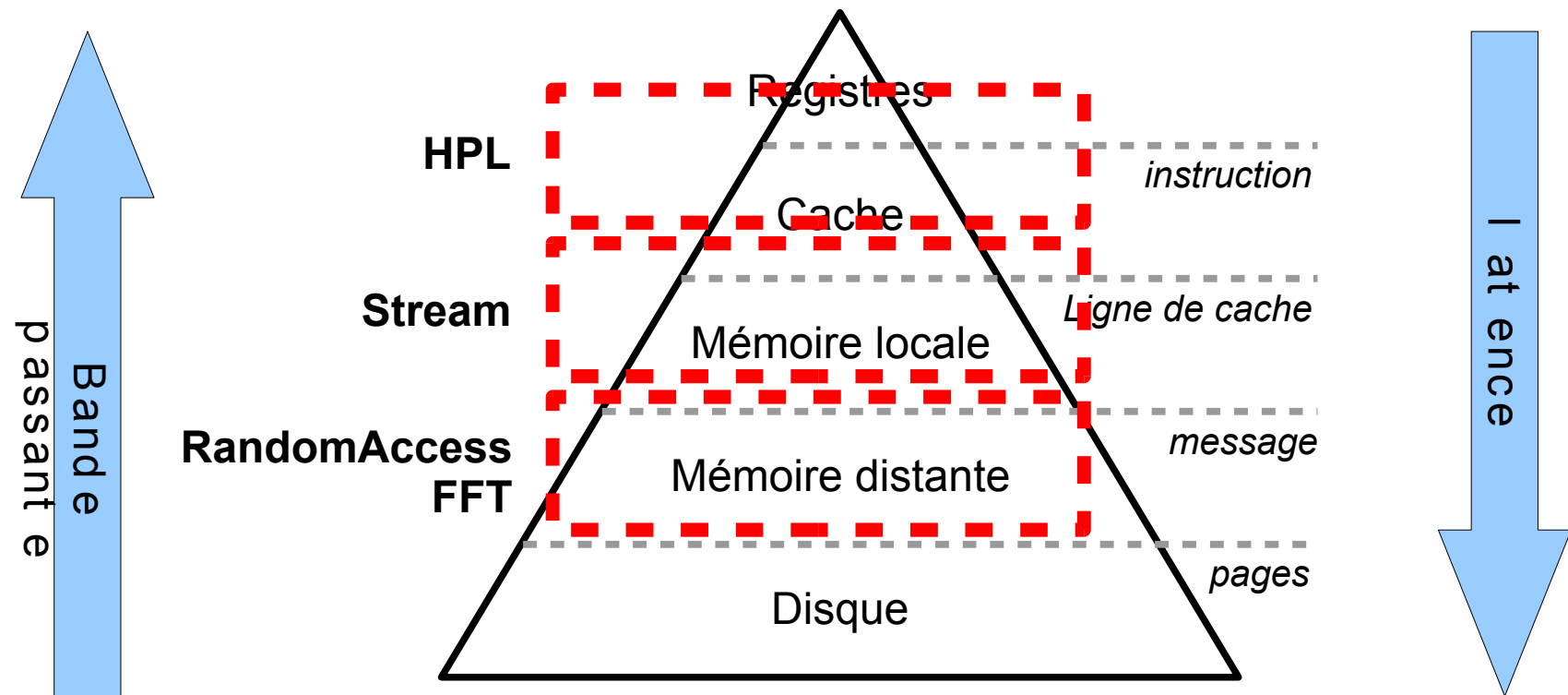


Type de runs

- Chaque code (s'il le permet) effectue 3 tests différents :
 - « single »
 - 1 seul processus
 - « embarrassingly parallel »
 - Tous les processus indépendamment
 - « global »
 - Tous les processus partagent la charge



Et ça teste quoi ?

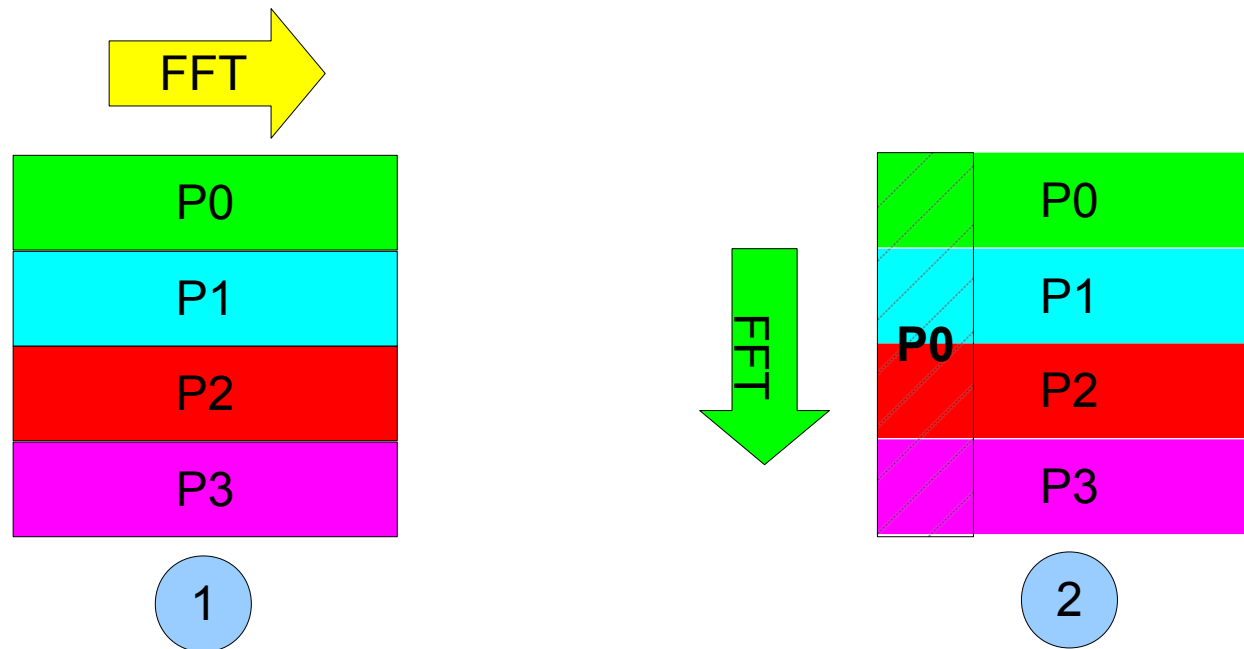


– RandomAccess

- Accès mémoires (quasi-) aléatoires
- Mise-à-jour d'emplacements (lecture-écriture) mémoires indexée par un tableau d'indices généré par un polynôme
- Si les accès sont aléatoires, les résultats devraient être comparables à un memlat :
- En réalité, sous-estime la latence (2-2.5).
Unité : Gups/sec = 1/latence (nanosecondes)

```
Ran = 1;
for (i=0; i<4*N; ++i) {
    Ran= (Ran<<1) ^ (((int64_t)Ran < 0) ? 7:0);
    Table[Ran & (N-1)] ^= Ran;
}
```

- Opération utilisée couramment.
- Le calcul distribué d'une FFT multidimensionnelle est « embarrassant » ; il requiert une transposition à chaque dimension qui induit une opération collective de type `MPI_alltoall(v)`.



- « matrix Parallél TRANSpOSE »
- Mise-à-jour du contenu d'une matrice par la somme de sa transposée et d'une matrice auxiliaire.
$$A = A^T + B$$
- La distribution des blocs est la même que HPL. Le découpage est important.
- On peut ajouter des cas supplémentaires dans le fichier de configuration.
- Le résultat est une bande passante (GB/sec)

– Matrice 9x9

- Découpage 3x3
 - 3 paires communicantes
- Découpage 1x9
 - 36 paires communicantes !

1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6
7	8	9	7	8	9	7	8	9
1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6
7	8	9	7	8	9	7	8	9
1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6
7	8	9	7	8	9	7	8	9

Découpage 3x3

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

Découpage 1x9

- Test l'interconnect (latence et débit) à l'aide de différents «pattern » de communications :
 - PingPong, natural (ordered) ring, random ring
 - Opérations uniquement point-à-point
 - Latence : 8 Bytes, Bande-passante 2 Mbytes
 - Rapporte pour chaque mesure le maximum, minimum et la valeur moyenne.
 - Attention à ce que l'on mesure : si l'on souhaite les débits inter-nœuds, penser à exécuter avec un seul processus par nœud

Beff

* Typical output on a Cray T3E:

* -----

*

* -----

* Latency-Bandwidth-Benchmark R1.5 (c) HLRS, University of Stuttgart

*

* Major Benchmark results:

* -----

*

* Max Ping Pong Latency: 0.005209 msec

* Randomly Ordered Ring Latency: 0.007956 msec

* Min Ping Pong Bandwidth: 314.025708 MB/s

* Naturally Ordered Ring Bandwidth: 147.600097 MB/s

* Randomly Ordered Ring Bandwidth: 61.096556 MB/s

*

* -----

*

* Detailed benchmark results:

* Ping Pong:

* Latency min / avg / max: 0.004268 / 0.004588 / 0.005209 msec

* Bandwidth min / avg / max: 314.026 / 318.653 / 324.822 MByte/s

* Ring:

* On naturally ordered ring: latency= 0.008512 msec, bandwidth= 147.600097 MB/s

* On randomly ordered ring: latency= 0.007956 msec, bandwidth= 61.096556 MB/s



Conclusion

Name	Generation	Computation	Communication	Verification	Per-processor data
HPL	n^2	n^3	n^2	n^2	p^{-1}
DGEMM	n^2	n^3	n^2	1	p^{-1}
STREAM	m	m	1	m	p^{-1}
PTRANS	n^2	n^2	n^2	n^2	p^{-1}
RandomAccess	m	m	m	m	p^{-1}
FFT	m	$m \log_2 m$	m	$m \log_2 m$	p^{-1}
b_eff	1	1	p^2	1	1

NAS Parallel Benhmarks

LIBERATE IT

NAS Parallel benchmarks

- <http://www.nas.nasa.gov/Resources/Software/npb.html>
- NASA Advanced Supercomputing (NAS) Division
- 8 benchmarks, essentiellement des noyaux couramment utilisés pour la résolution de problèmes en dynamique des fluides
 - Embarrassingly parallel (EP)
 - Multigrid (MG)
 - Conjugate gradient (CG)
 - 3-D FFT PDE (FT)
 - Integer sort (IS)
 - LU solver (LU)
 - Pentadiagonal solver (SP)
 - Block tridiagonal solver (BT)

NAS Parallel benchmarks

– Portage

- Fortran90 et C
 - Nécessite une implémentation MPI
 - Simple
 - Adapter le fichier ../config/make.def
 - Makefile
- ```
make <benchmark-name> NPROCS=<number> CLASS=<class>
```

## – Exécution

- Simple !
- Autovérifiant
- Les sorties, complètes, fournissent un certain nombre de métriques, dont des MFLOPS/sec