Pot pourri

PAP en 15 transparents ;-)

Équilibrage de charge

- Problèmes réguliers
  - Calcul prévisible ne dépendant pas des résultats intermédiaires
    - Ex. Algèbre linéaire classique
    - Une distribution équitable *a priori* est possible
      - De façon statique, à la compilation
      - Juste avant l'exécution
    - Problème NP-Complet (analogue au *bin packing*)
- Problèmes irréguliers
    - Ex. calcul fractal, décomposition en facteur premier
    - Il faut rééquilibrer dynamiquement la charge au fur et à mesure

Loi d'Amdahl

$$\text{accélération} = \text{speedup} = \frac{\text{temps du meilleur prog. séquentiel}}{\text{temps du prog. parallèle}}$$

Soit *S* la part séquentielle de l'application A pour une donnée d. L'accélération de l'exécution de A(d) sur une machine à p processeur est bornée par

$$speedup(p) \leq \frac{1}{S + \frac{1-S}{p}}$$

« si 1% de l'application est séquentielle on n'arrivera pas à aller 100 fois plus vite »

Jeu de la vie
Encore plus de désynchronisation

- Réduire le nombre de synchronisations
  - On peut calculer l'état d'une cellule sur k étapes si on connaît l'état des cellules à distance k.

  - Idée : remplacer des synchronisations par du calcul redondant

Conclusion sur le chapitre

- Programmer avec les threads
  - C'est bien car
    - On contrôle tout
    - On peut inventer ses propres mécanismes de synchronisation
  - Mais c'est un peu pénible…
    - Surtout pour un non informaticien
    - Souvent les mêmes schémas
    - Modification lourde du code
- Synchronisation / calcul redondant / Mémoire
  - Un compromis subtil

Hello world !

```
#include <omp.h>

int main()
{
#pragma omp parallel
  printf("bonjour\n");

 printf("au revoir\n") ;

  return EXIT_SUCCESS;
}
```

```
> gcc –fopenmp bon.c
>OPENMP_NUMTHREADS=3 ./a.out
bonjour
bonjour
bonjour
au revoir
>
```

Tâches

```
#pragma omp parallel // intialisé un sac de tâches pour l'équipe
{
…
#pragma omp task // initier une tâche, créer un sac pour ses sous tâches
{
    …
#omp taskwait // attendre la fin de toutes les tâches créées dans cette tâche
    ….
}

#pragma omp taskyield // passer la main à une autre tâche

…
#omp taskwait // attendre la fin de toutes les tâches créées dans le bloc
…
}
```

- Moyen de générer du parallélisme à grain fin
  - Surcoût moindre en comparaison à celui des threads
  - Simplifie parfois la programmation

TBB –TSP
Cut-off

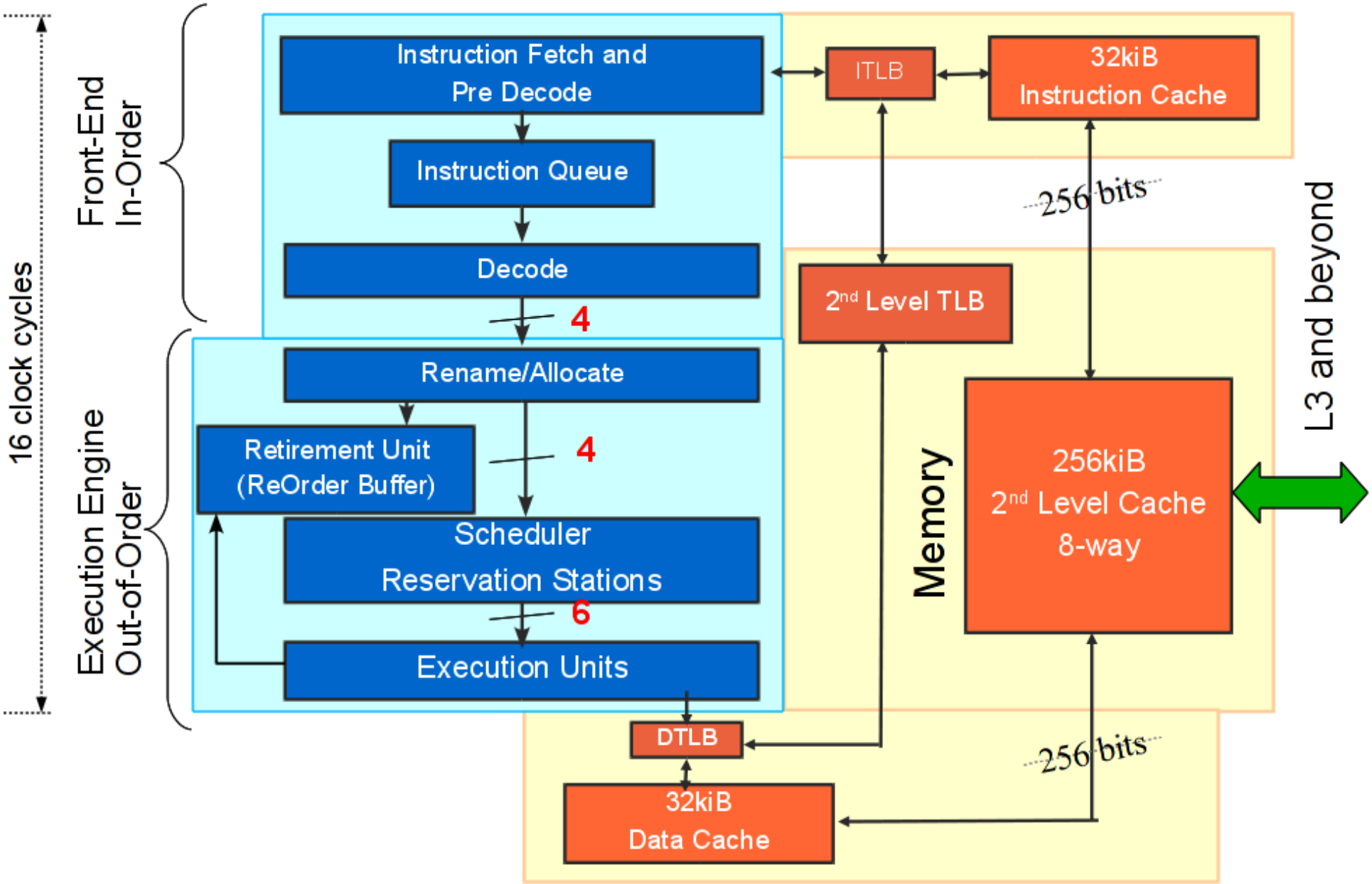
Passer sur code séquentiel à partir d'un seuil

Version	15 Städte	16 Städte	17 Städte
Sequentiell	5,75 s	30,68 s	130,46 s
task + cut	0,58 s	3,26 s	14,18 s
Speedup	9,91	9,41	9,20

Speed-up super linéaire

Cut-Off	Laufzeiten in Sekunden		
	15 Städte	16 Städte	17 Städte
1	6,25	32,20	133,09
2	1,06	5,06	19,59
3	0,71	4,14	15,70
4	0,64	3,26	14,86
5	0,62	3,49	14,18
6	0,58	3,55	15,29
7	0,65	3,66	16,21
8	0,73	3,83	15,00
9	0,94	4,64	18,75
10	1,34	6,47	22,42
11	2,09	8,39	32,05
12	2,14	10,29	38,96
13	1,99	10,68	40,78
14	2,09	11,14	45,90
15		10,98	44,10
16			48,31

Nehalem Core Pipeline



Cache
Optimisation des programmes

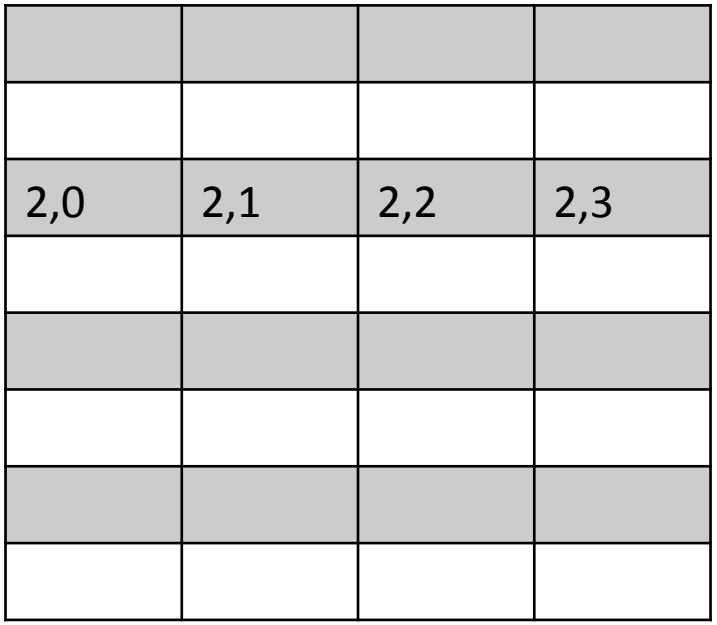
- Parcours séquentiel d'un tableau

```
#define N 8192
typedef long matrix[N][N];

 for (j=0;j<N;j++)
     for (i=0;i<N;i++)
        C[i][j] = A[i][j] + B[i][j];
```

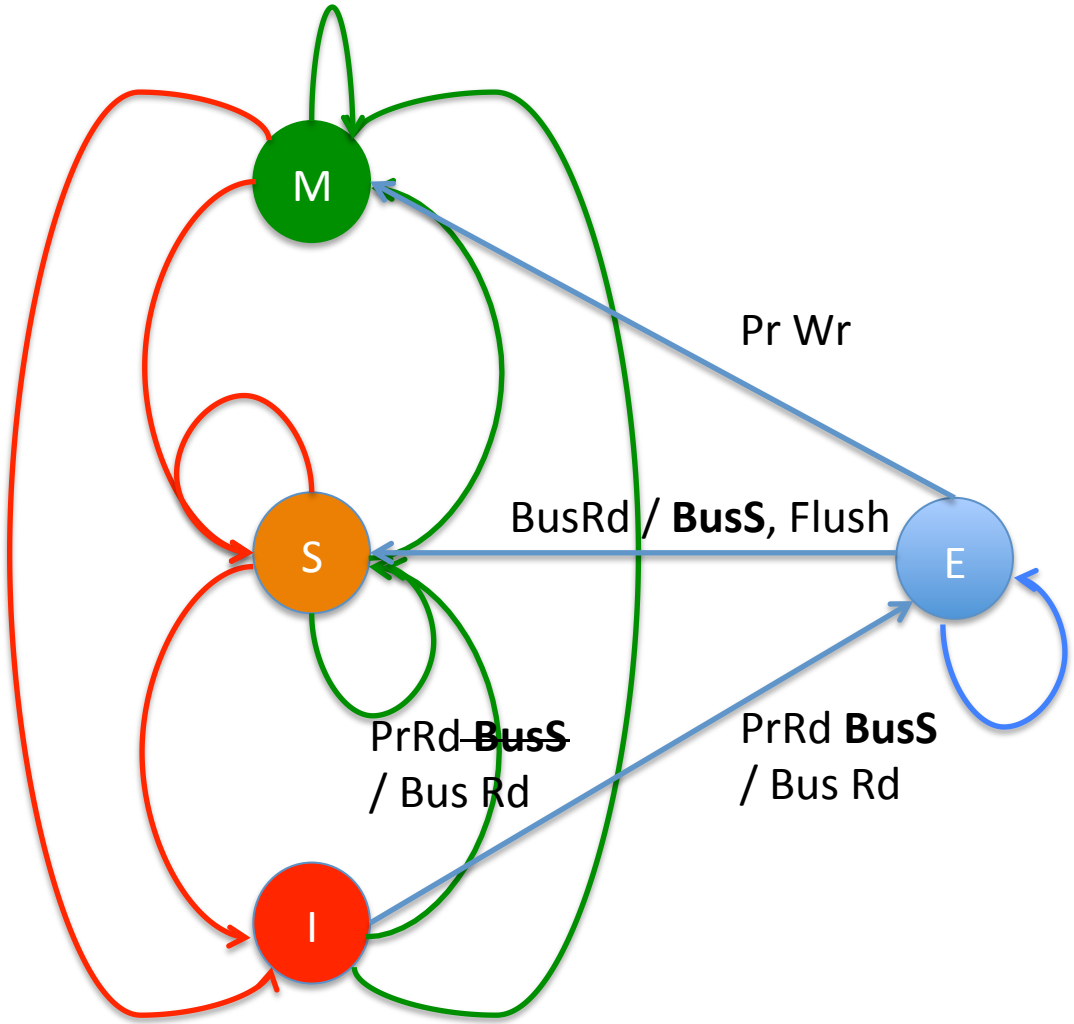
Contre productif :

- 1 défaut de cache par lecture
- Et rapidement 1 défaut de TLB par lecture



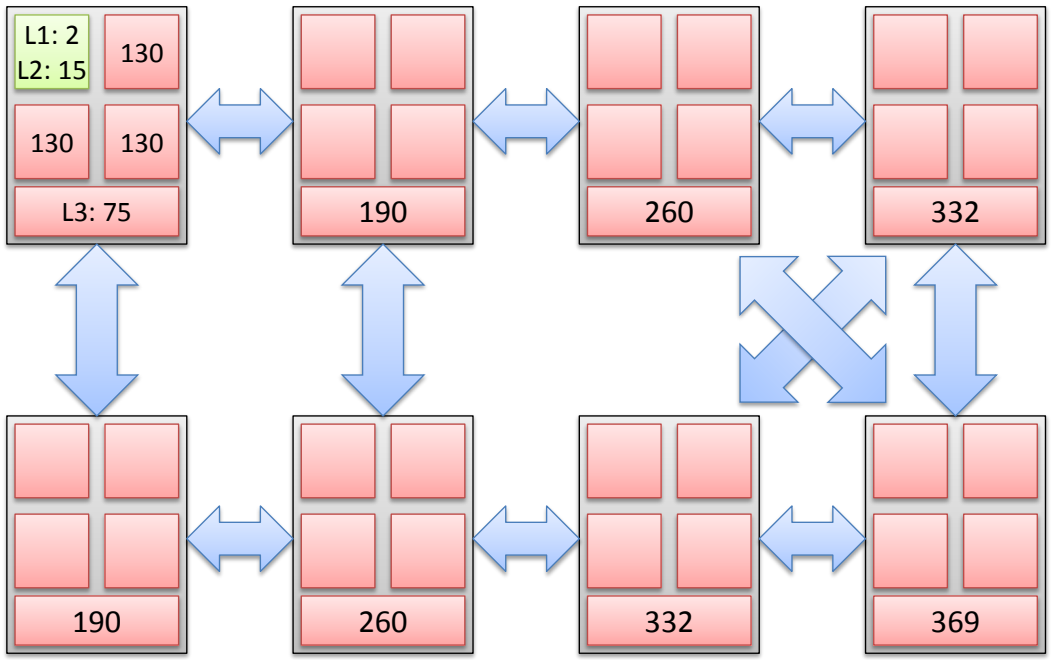
Protocole MESI
Modified Exclusive Shared Invalid

- Transition silencieuse si variable non partagée
  - Nécessite de savoir si la valeur provient d'un cache ou de la mémoire



Opteron 8 sockets

Cache access latency



Memory is a bit faster than L3, but it's complicated…

TP1 sur Boursouf
initialisation séquentielle

	1	2	4	8	16	32
addseq:	1095	1152	1053	1151	1152	1149
addparastat:	1703	965	909	605	654	632
addparadyna:	1698	794	746	549	564	535
sumseq:	237	239	239	239	239	238
sumparastat:	5167	43172	43798	23559	16465	14974
sumparapart:	340	275	275	195	202	207
addparastat:	0,64	1,19	1,16	1,90	1,76	1,82
addparadyna:	0,64	1,45	1,41	2,09	2,04	2,15
sumparastat:	0,05	0,01	0,01	0,01	0,01	0,02
sumparapart:	0,70	0,87	0,87	1,22	1,18	1,15

TP1 sur Boursouf
initialisation parallèle

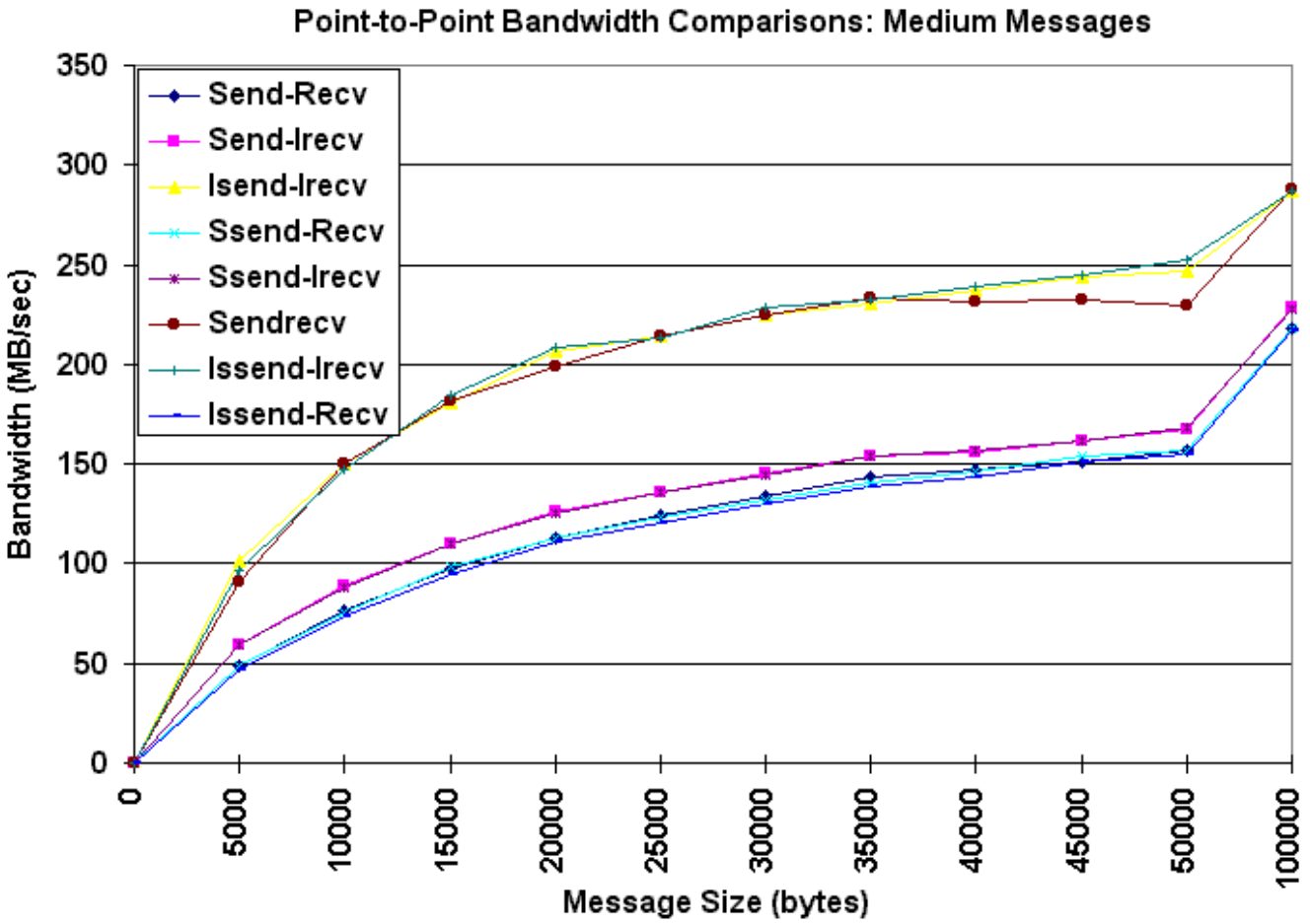
	1	2	4	8	16	32
addseq:	1616	1794	1897	1889	1884	1878
addparastat:	956	510	260	132	87	57
addparadyna:	960	845	615	505	499	425
sumseq:	379	459	502	481	480	477
sumparastat:	4430	7828	37887	21038	15564	13635
sumparapart:	222	112	56	28	17	23
addparastat:	1,14	2,26	4,05	8,72	13,24	20,16
addparadyna:	1,14	1,36	1,71	2,28	2,31	2,70
sumparastat:	0,05	0,03	0,01	0,01	0,02	0,02
sumparapart:	1,07	2,13	4,27	8,53	14,05	10,36

Paradigmes de la programmation des architectures distribuées

- Approches explicites
  - Passage de message
    - Programmation à base de Send / Receive
  - appel de procédure à distance
    - Modèle client / serveur
    - Programmation à base de RPC,
      - Java JEE (RMI, Corba), WebService,…
- Approches implicites
  - « mémoire virtuellement partagée »
  - « système distribué à image unique »

Performances (LLNL)

https://computing.llnl.gov/tutorials/mpi_performance/



Recouvrement calcul communication comportement avec copie