Les temps de restitution ...

LIBERATE IT

Les temps de restitution

- Tout le monde est pressé et veut des résultats tout de suite
- Un code est-il performant parce qu'il tourne vite ?
- Code parallèle : la parallélisation sert-elle uniquement à faire à ce que le code donne un résultat plus rapidement ?

Performance brute des codes séquentiels

Code	cas test	W 3.0 GHz	CL 2.4 GHz	CI/WC
A	1	2	2	37%
	2	30	41	40%
	3	2	3	42%
	4	44	60	38%
	5	155	207	34%
B	1	178	193	8%
C	1	79	98	24%
	2	4	4	-5%
	3	6812	8410	23%
D-1	4	645	807	25%
D-2	5	704	862	22%

- Lecture brute : avantage au woodcrest

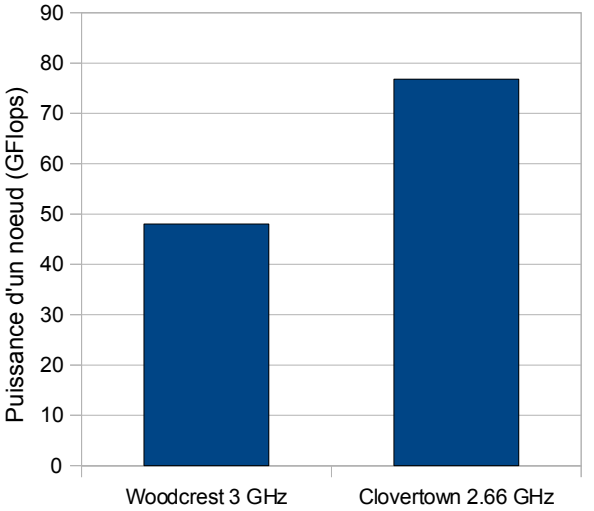
Effet de la charge

Code	cas test	W 3GHz	CL 2.4 GHz	CI/WC	W 3GHz	CL 2.4 GHz
A	1	2	2	0%	33%	-3%
	2	64	162	153%	117%	293%
	3	5	10	100%	127%	219%
	4	99	250	153%	127%	314%
	5	347	890	156%	124%	329%
B	1	291	580	99%	63%	201%
C	1	106	181	71%	34%	84%
	2	4	5	25%	7%	40%
	3	11760	26070	122%	73%	210%
D-1	4	702	818	17%	9%	1%
D-2	5	788	881	12%	12%	2%

- Lecture brute : la charge dégrade les performances, les codes sont collés la bande-passante, mais reste l'avantage au woodcrest

Vue globale de la machine

Code	cas test	W 3GHz	CL 2.4 GHz	
A	1	43200	86400	100%
	2	1350	1066	-21%
	3	17280	17280	0%
	4	872	691	-21%
	5	248	194	-22%
B	1	296	297	0%
C	1	815	954	17%
	2	21600	34560	60%
	3	7	6	-14%
D-1	4	123	211	72%
D-2	5	109	196	80%

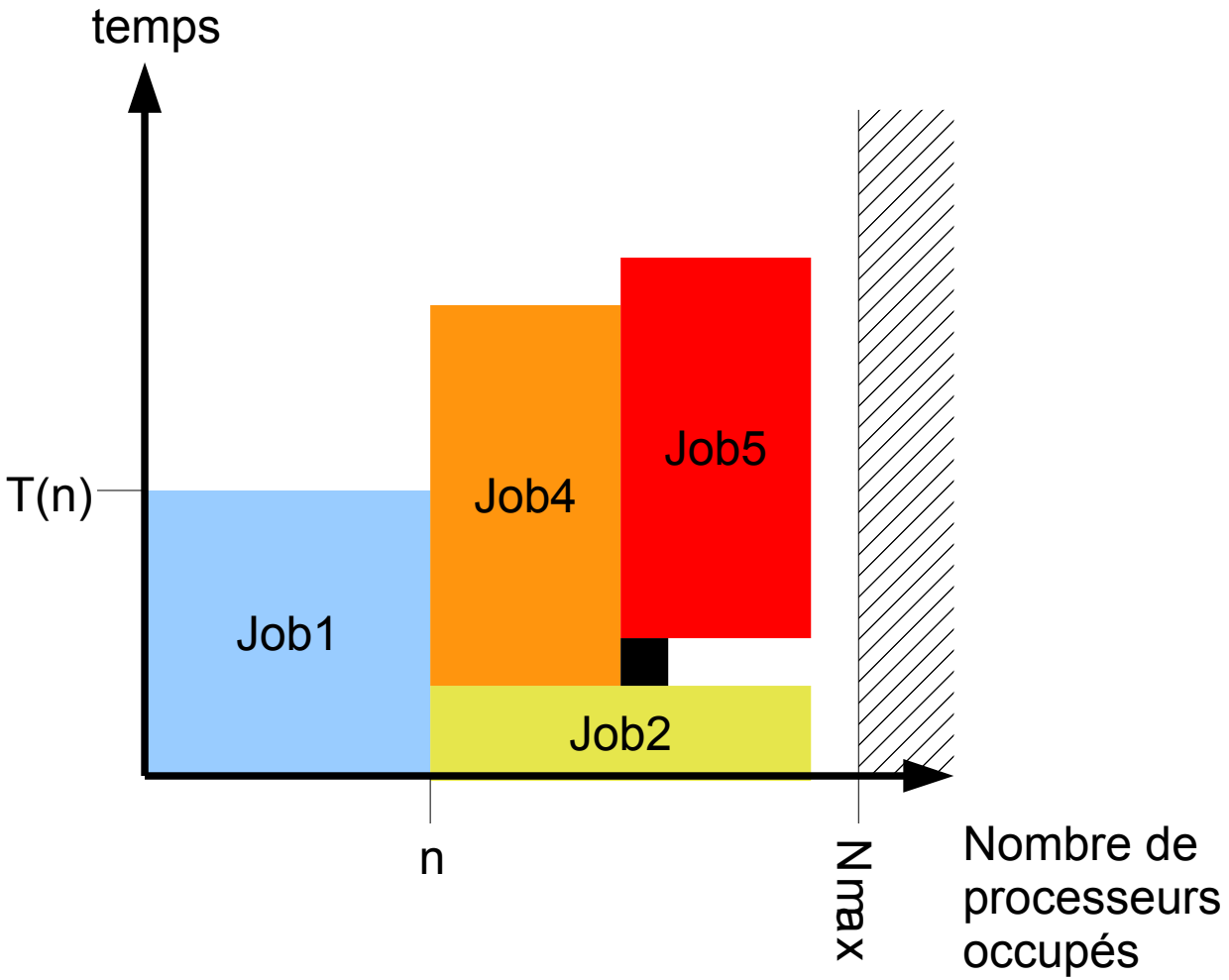


- Le prix de processeurs Clovertown 2.4 GHz était le même que celui des Woodcrest 3.0 GHz : on travaille à nombre de noeuds constant et non nombre de coeurs constant!

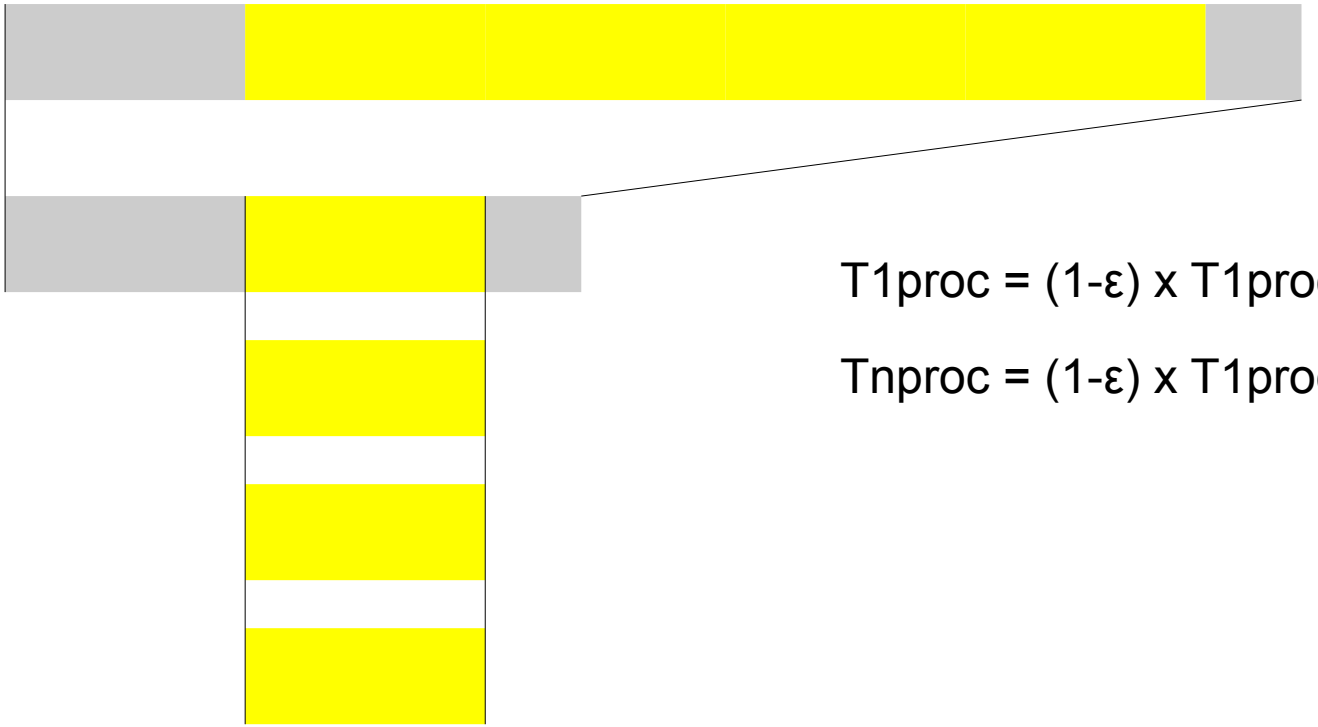
En conclusion

- La charge détériore les performances (stress de la bande passante)
- Il faut faire un compromis entre performance individuelle des codes et performance globale de la machine.
- Attention l'analyse à la feuille excel :
  - Ce type d'analyse s'explique dans un rapport de benchmark pas dans une cellule de votre tableur favori !
  - Il ne faut pas passer à côté de quelque chose !
- Il faut toujours voir une machine et son utilisation sous ces deux dimensions : le nombre de cœur et le temps (queue d'occupation d'un « gestionnaire de batch »).
  - Le nombre maximum de processeur (axe des X) est fini : il est contraint par des aspects financiers, de place, de puissance électrique, de puissance dissiper. Les benchs aide au sizing de cette quantité.
  - Les temps de restitution des jobs ne sont que des « boites » qui remplissent cet espace.

Vision à 2D



Loi d'Amdhal



$$T1proc = (1-\varepsilon) \times T1proc + \varepsilon \times T1proc$$

$$Tnproc = (1-\varepsilon) \times T1proc + \varepsilon \times T1proc / n$$

- Loi d'Amdhal :
  - Déduire la part séquentielle, la part parallélisable
  - A très grand nombre de processeurs, le temps global de restitution temps vers le temps séquentiel ...
    - L'efficacité d'un code parallèle tient dans sa capacité à pouvoir réduire sa part séquentielle ...

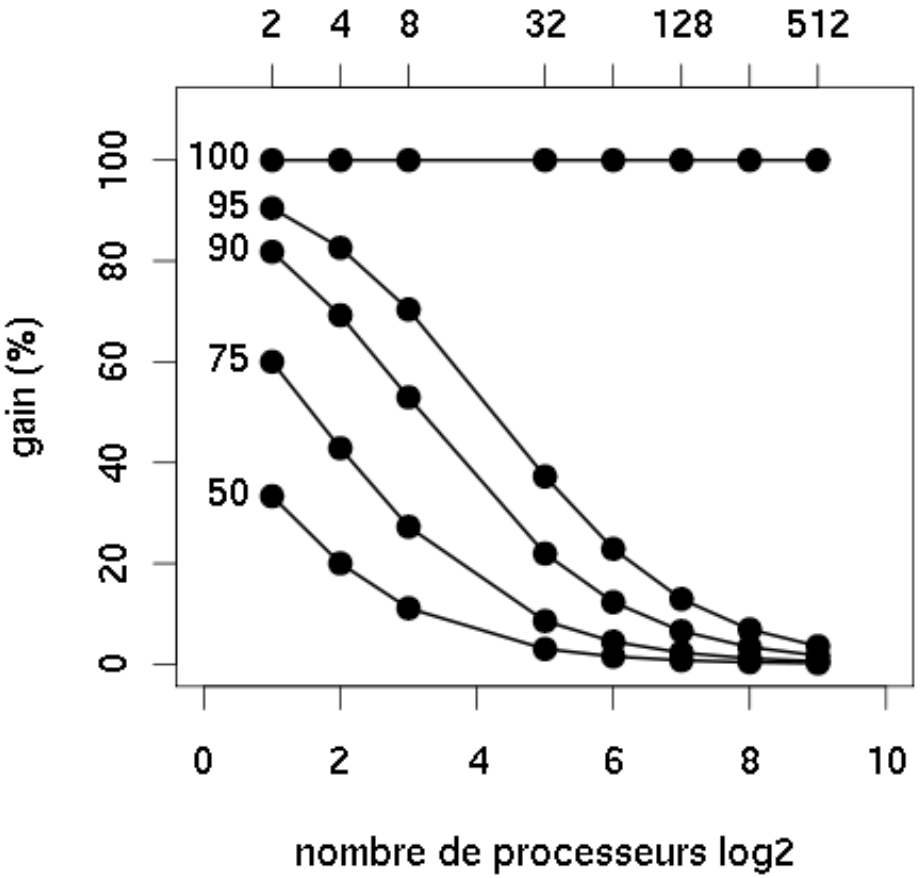
Critique de la loi d'Amdhal

- Il y a plusieurs problèmes à cette loi :
  - Son domaine d'application est limité
    - On néglige tous les terme d'ordre supérieur à 1 dont les communications
      - → Négliger les communication n'est pas acceptable pour un code parallèle (en général!)
      - → A grand n sont sont ces termes qui domine vis-à-vis de 1/n ... la loi est dure mais c'est la loi ...
  - Les parties séquentielles (IO, initialisation) ne sont pas forcément elles-mêmes indépendante de n !
  - Il implique de l'utilisateur se focalise à **ressources constantes** sur la diminution sans condition de son temps de restitution, mais est-ce bien raisonnable ???
    - Argument de John L. Gustafson : « *it may be most realistic to assume that run time, not problem size, is constant* »
    - Maillage de 1000x1000x10000, en double précision représente 74 GiB de mémoire !
      Sur des machine de 8 coeurs avec 2 GiB/coeurs : au minimum 5 machines !!!

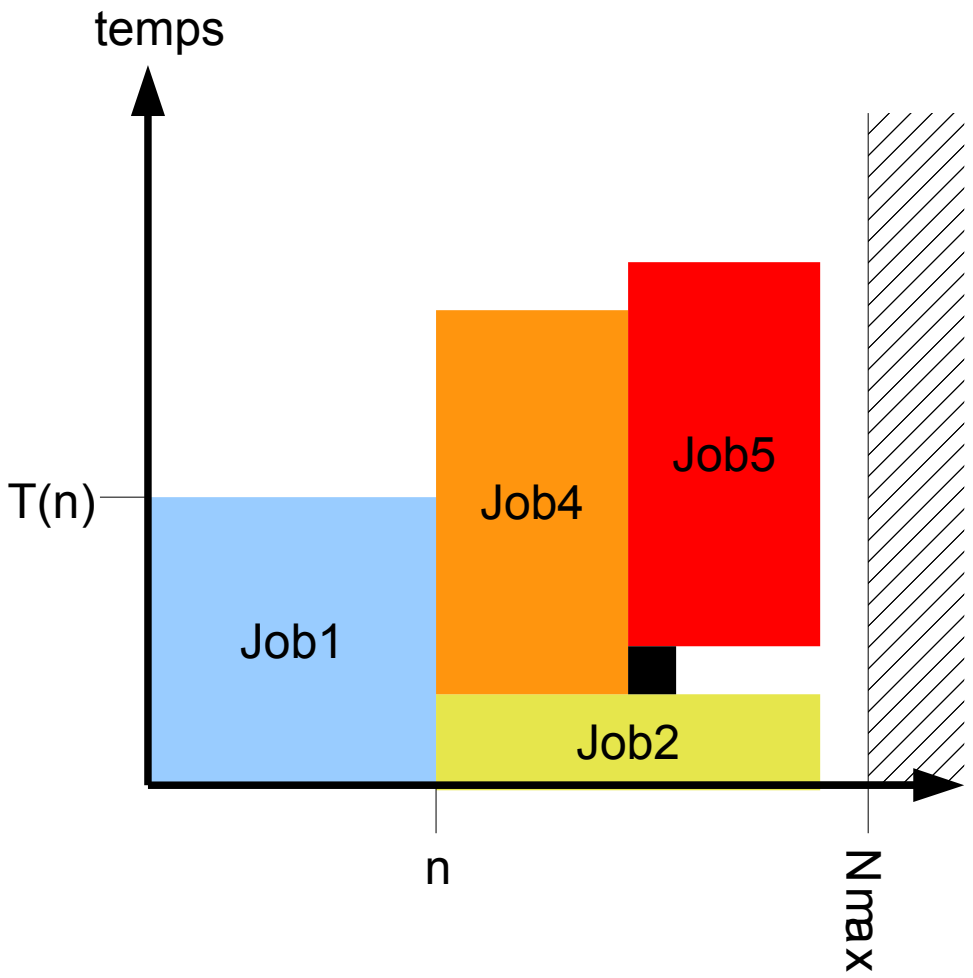
Passage à l'échelle

- Un paramètre important à déterminer dans le cas des applications parallèle est la « scalibilité »
  S(n) = T1proc/Tnproc
- Si l'on suit la loi d'Amdhal, elle temps vers 1/(1-ε) pour les n grands.

Scalabilité

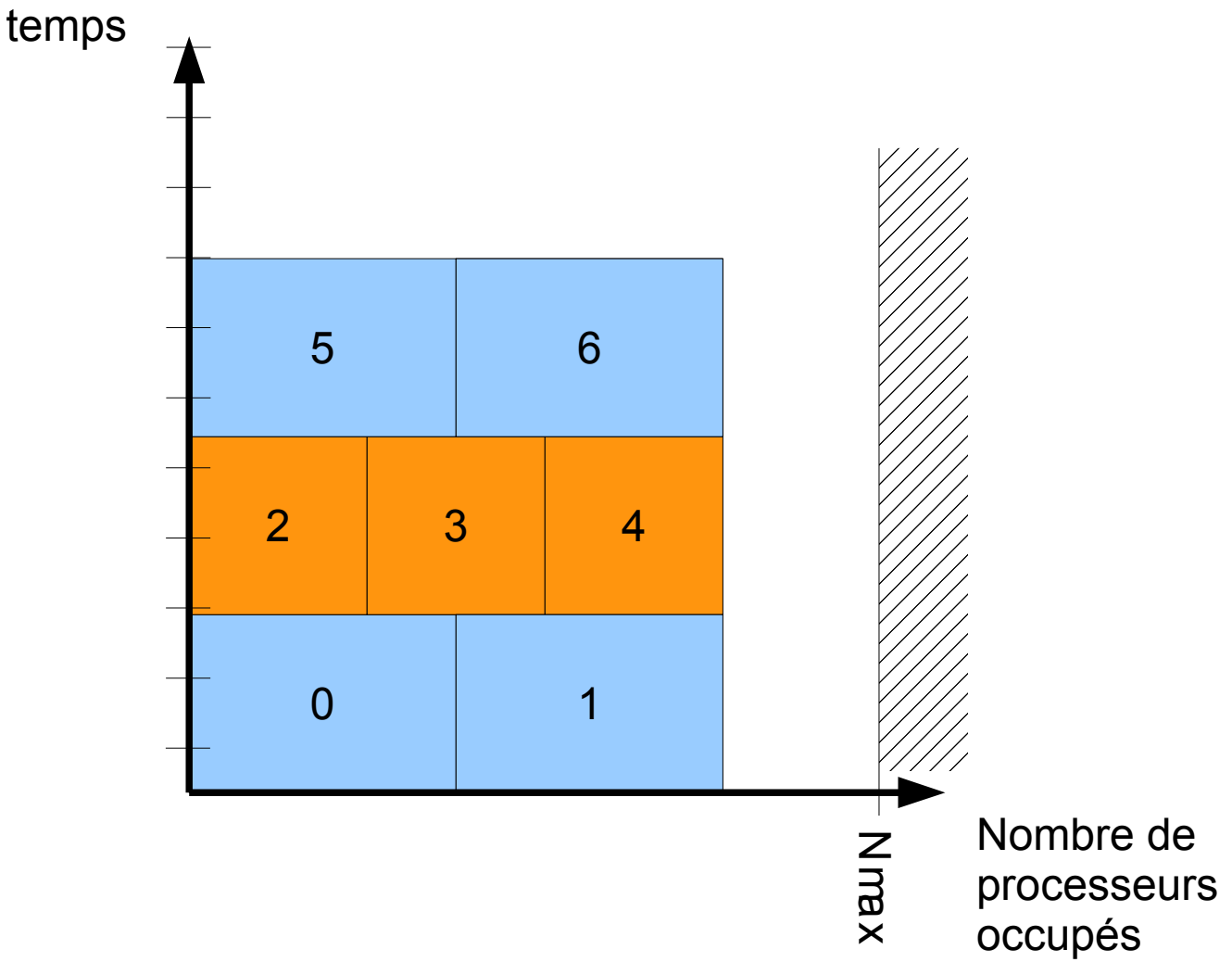


N

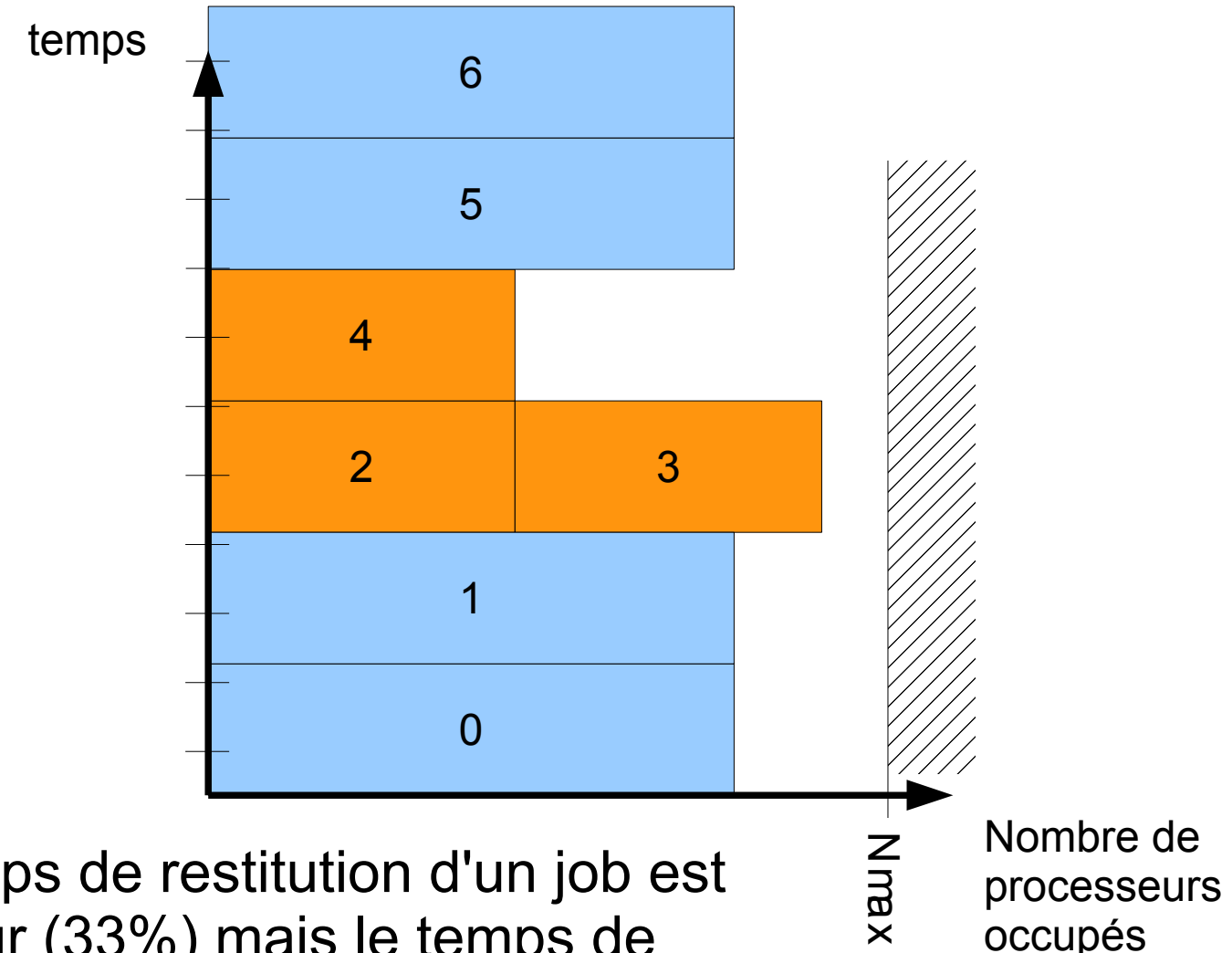


- Surface d'une boîte :
  $S(n) = n\ T(n) = n(1-\varepsilon)T1 + \varepsilon T1$
- L'idéal c'est S(n) soit constant avec n
  - Vrai si ε=1
- Problème :
  - L'utilisateur est tenté de réduire ces temps de restitution
    - Augmenter n !

N



N



- Le temps de restitution d'un job est meilleur (33%) mais le temps de restitution de n jobs c'est fortement dégradé ...
- Il faut un compromis

Préparer ces benchmarks

LIBERATE IT

Faire son choix de benchmarks

- Un résultat de benchmark est donné pour un code et son cas test
  - Adhérence au cas test
  - Plusieurs cas tests, de tailles différentes
  - Peut-être voulu (différent type d'exploitation d'un code en production)
    - Un cas test peut-être fait pour tester tel ou tel élément architectural
      - → IO
      - → Cache
      - → Mémoire
      - → Interconnect ...
- Penser quels paramètres architecturaux peuvent influer sur mes benchmarks

Faire son choix de benchmarks applicatifs

- Le choix doit être cohérent et surtout représentatif avec la production.
  - Impliquer les utilisateurs
- Tenter d'isoler des utilisations récurrentes communes et extraire des benchmarks simples
  - Par exemple utilisateurs de
    - BLAS/CBALS/LINPACK
    - FFTW
- Réduire les cas tests
  - Taille des données, nombre d'itération, temps de restitution.
  - Attention : réduire un cas test peut changer le comportement de l'application ! (penser aux effets de cache)
  - Compromis
- ISV
  - Pb des licences

Une machine pourquoi faire

- But

  - **Ma machine sera-t-elle**
    - Dédiée à une application ?
    - Destinée à des applications ayant de forts points communs
      - → Ex. Mécanique des fluide → FFT ?
    - Applications hétérogène avec des profils utilisateurs totalement différents ?

  - **Une machine pour**
    - absorber de la charge : job mono cpu, openMP, petits jobs MPI ?
    - Exécuter des gros jobs MPI, sur un grand nombre de nœuds avec la meilleure scalabité possible (si le code s'y prête), et utilisant beaucoup de ressource
    - Bon interconnect, réseau non-bloquant, machine avec beaucoup de mémoire
    - Et tout le reste entre les deux !

Quel travail demander ?

- Temps de restitution
  - De jobs indivuels :
    - Jobs mono cpu, open MP
      - → Attention au effet de charge !
        - Job avec 1 PPN et noeud plein
    - Jobs MPI : courbe de scalabilité
  - Charge : « combien de temps faut-il pour tourner une charge donnée ? »
- Sizing de configuration
  - Sans la contrainte budgétaire, avec ...
  - Job individuel : « soit mon code, connaissant ces caractéristiques, quelle taille de machine pour m'assurer d'avoir un résultat au bout de 12 heures ? »
  - Charge : « Quel nombre de nœuds pour tourner une charge donnée sur 24 h ? »
  - Attention, Extrapolations !
- Il faut être raisonnable dans ces choix

Formaliser votre demande de benchmarks

- Les règles de benchmarks et conditions de passage devront être clairement exprimés dans le CCTP
  - Code, nombre de nœuds, en charge ou pas, droit aux modification de code ...
    - Il vaut mieux toujours demander les résulats des codes « as-is » même si l'on autorise les modifications ...
  - Hardware : Hyperthreading ? Turbomode ? ...
- Vous devrez fournir en plus du CCTP :
  - Les codes et les cas tests (tar.gz, md5sum)
    - Attention à certain ISV ou codes universitaires sous licence ...
    - Pour les code « tiers » (ISV et autres), indiquer les versions désirées
  - Un readme clair par code
    - Rappel des conditions de passage
    - Indication sur le portage :
      - → Dépendances sur des biliothèques (Préciser les versions)
    - D'éventuels timings
    - Des fichiers de référence (indispensables)
      - → Je ne suis pas un spécialiste de la chimie moléculaire ...

Formaliser votre demande de benchmarks

- Le travail de benchmarking même côté client est un gros travail : équipe d'utilisateurs et de gens plus proches du système
  - Point de contacts
- Préciser les sorties qui vous seront nécessaires
- Éviter les scripts qui font tout ...
  - Vous ne connaissez pas l'environnement du vendeur
  - C'est un environnement de production
  - Dans 80% on fait autrement

En retour

- Le délivrable : le rapport de benchmark.
  - Document à part entière ou intégré « physiquement » dans la réponse commerciale
- Un « *executive summary* » reprenant les engagements sur la machine cible.
- Un descriptif clair de la machine
  - Cible
  - De benchmarks
    - Hardware
    - Pile software y compris compilateurs, bibliothèques ...
- Pour chaque code
  - Les engagements, les mesures, une analyse et l'explication de la méthode d'extrapolation s'il y a lieu.
  - Le mode opératoire : portage, compilation, execution
    - Descriptif des modifications de code
    - Options de compilation

En retour

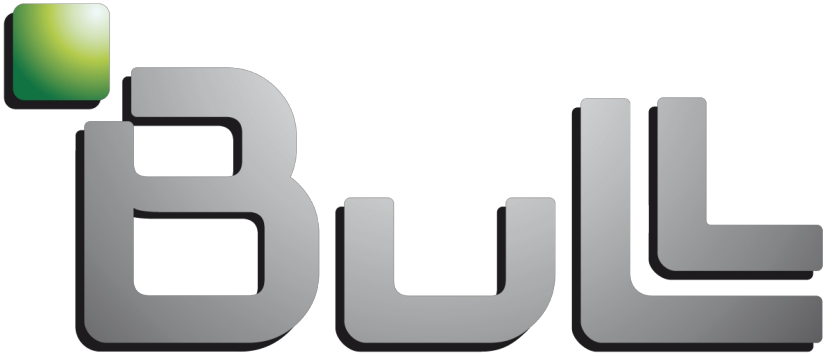
- Un fichier contenant tous les sorties désirées, les codes modifiés s'il y a lieu.
- Ce ne sont pas des règles universelles : c'est ce qui ME semble important. Ne pas hésiter à préciser toute choses qui vous paraissent indispensables bien qu'elles vous paraissent évidente.

Extrapolation

- Pourquoi extrapole-t-on ?
  - Ressources limités
    - Hébergé des ressources à un coup (immobilisation de matériel, dépréciation)
      - → Nombre de cœurs limité
  - Evolution technologique
    - Les technologies dans le HPC évoluent vite
      - → Le modèle du « tick-tock » d'Intel est de deux ans
    - De l 'émission de l'AO à la livraison, plusieurs mois (~1 an) s'écoulent
      - → Le vendeur propose des solutions en avance de phase, en accord avec sa roadmap et celle de ses fournisseurs.
- Extrapolation : l'**art du benchmarkeur**
  - Du bon sens ! De la connaissance !
  - Elle doit être convaincante : modèles simples
  - La méthode doit-être expliquée

Conclusion

- Il n'y a pas de compétitions entre vendeurs et acheteurs : seulement entre vendeurs ...
  - Relation de confiance
- Vous n'achetez pas une machine pour exécuter des benchmarks
  - Ce sont des outils
- Vous n'achetez pas une machine pour faire tourner un HPL et/ou rentrer dans le top500
  - C'est juste la cerise sur le gâteau !
- Au final, vous devez satisfaire vos utilisateurs
- Au final, je dois satisfaire mon client



Architect of an Open World™

LIBERATE IT