Parallélisation d'opérateurs de TI: multi-cœurs, Cell ou GPU ?

Lionel Lacassagne[1], Antoine Pédron[2], Florence Laguzet[3], Michèle Gouiffès[1]

1 = Institut d'Electronique Fondamentale, Université Paris Sud
2 = Laboratoire de Dévelopement Informatique, CEA List
3 = Laboratoire de Recherche en Informatique, Université Paris Sud

Lionel.lacassagne@u-psud.fr

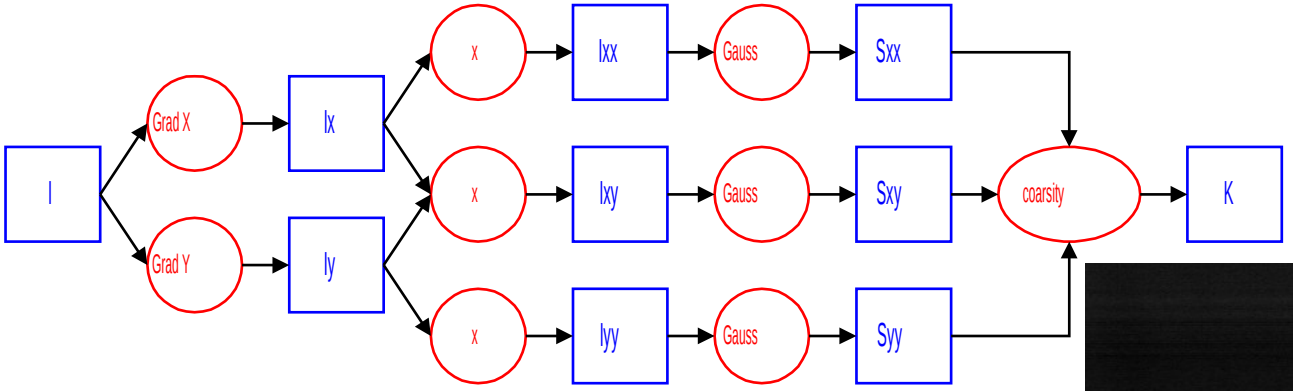
Contexte

- Calcul Haute Performance pour le traitement d'image: quelles sont les pistes actuelles ?

- Métrique de comparaison
  - Vitesse de calcul (respect de la cadence de traitement)
  - Consommation électrique (respect de la contrainte d’embarquablité)

- 3 modèles d’architecture
  - GPP RISC, multi-cœurs SIMD: parallélisation et vectorisation de code
  - GPU many cores: parallélisation massive
  - Cell: multi-cœurs à mémoire distribuée (DMA): modèles de parallélisation

Choix de l’algorithme de Harris



- Traitement d’image:
  - représentatif des algorithmes « bas niveau » (calculs régulier indépendant des données)
  - composé d’opérateurs ponctuels et de convolutions
- Architecture:
  - 4 étapes successives de calculs (source d'optimisations supplémentaires)

Transformations algorithmiques et optimisations logicielles

- Transformations algorithmiques haut niveau: filtre séparable
  - une convolution 2D = deux convolutions 1D

- Optimisations logicielles classiques
  - Déroulage de boucle, rotation de registre: diminution nb accès mémoire
  - entrelacement des données: diminution références actives (associativité cache)
  - Scalarisation : mémorisation valeurs dans registres

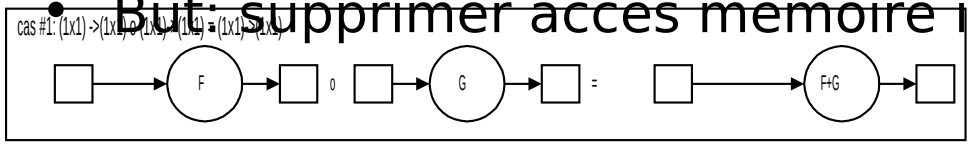
- Optimisations liées au domaine applicatif
  - Réduction par colonne (combinaison filtre 1D + déroulage de boucle) : diminution nb accès mémoire **et** nb calculs
  - La spécialisation des optimisations généralistes, appliquées au TI augmente leur efficacité

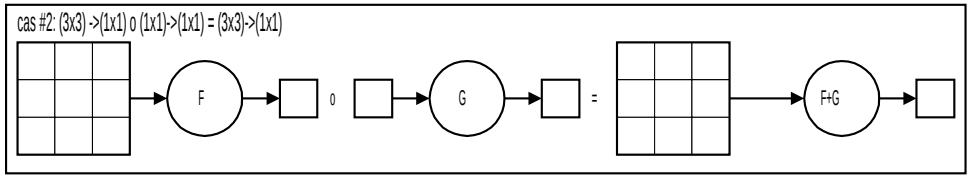
- Vectorisation de code
  - Utilisation des instructions SIMD (Altivec ou SSE)

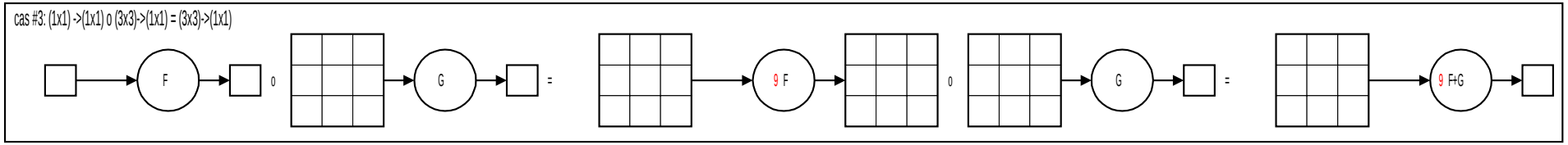
- Parallélisation de code (OpenMP)
  - Multi-threading sur processeur multi-coeurs

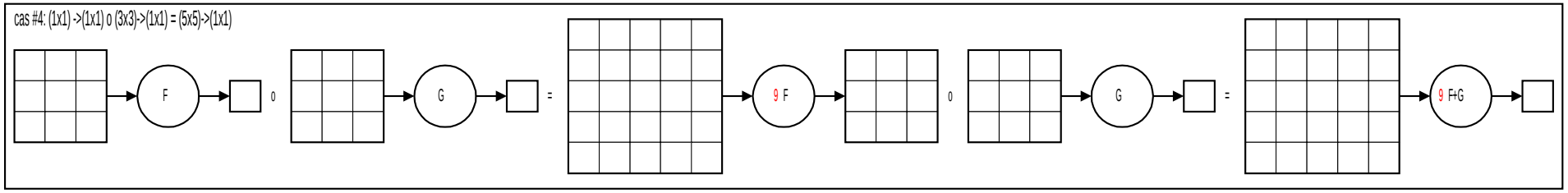
Transformations algorithmiques : fusion d'opérateurs

- Caractérisation des opérateurs via modèle producteur-consommateur
  - 1 motif de consommation de donnée et 1 motif de production de donnée
- Fusion d'opérateurs
  - Si motifs sont compatibles, sinon adaptation des motifs
- But: supprimer accès mémoire intermédiaires

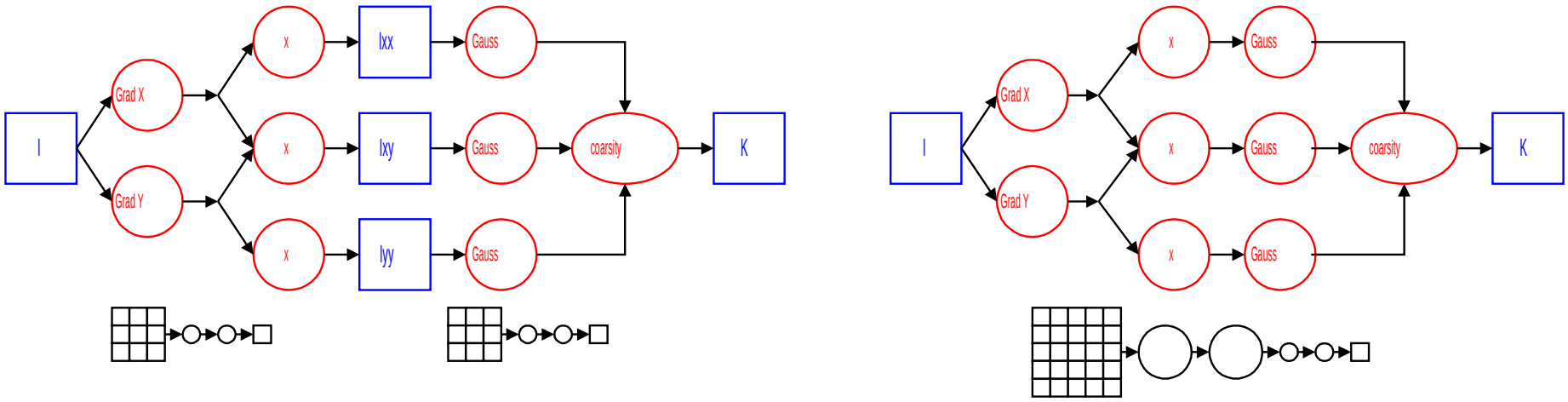








Transformations Halfpipe et Fullpipe



- Complexité et intensité arithmétique des versions *Planar, Halfpipe et Fullpipe*

version	Planar	Halfpipe optimisé	Fullpipe optimisé
LOAD + STORE	54+9=63	12+4=16	5+1=6
MUL + ADD	29+35=64	18+19=37	32+31=63
intensité arithmétique	1.02	2.31	10.5

- Diminution de la complexité, amélioration de l'intensité arithmétique
- *Memory bound* ou *computation bound ?*

Résultats et impact des optimisations

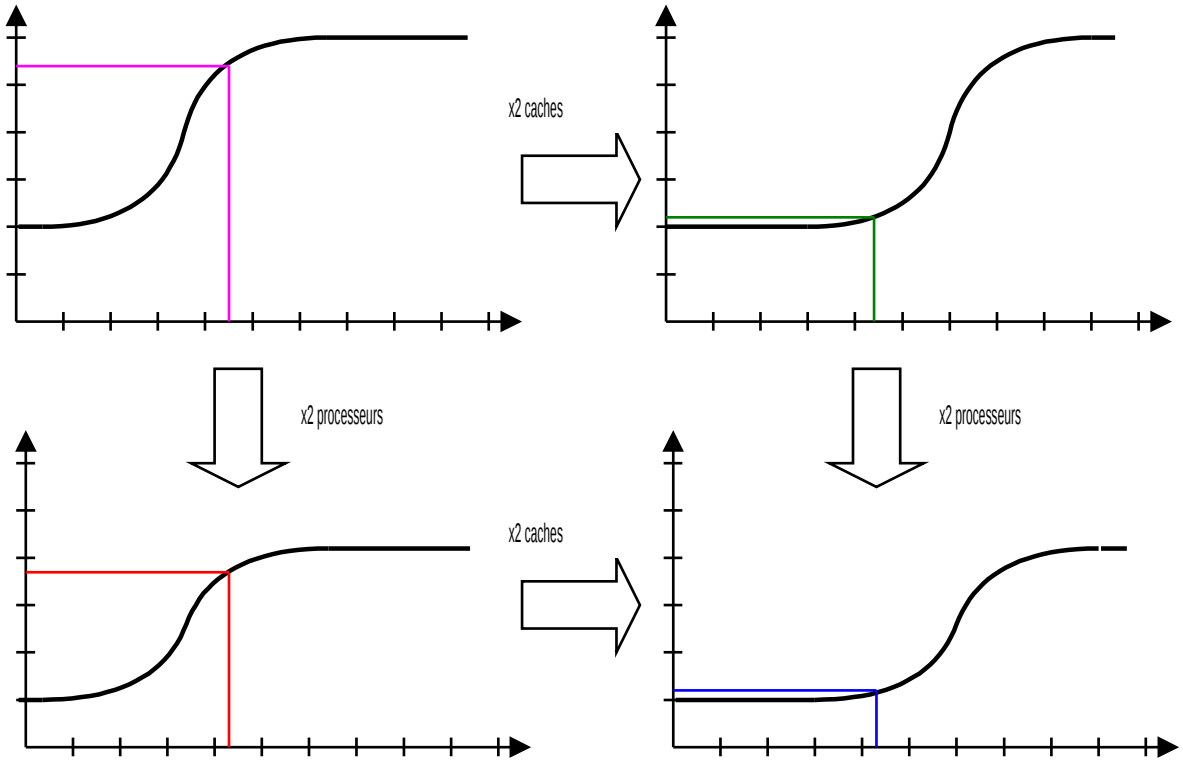
- Performances des versions Planar et Halfpipe (gain et temps de calcul)
  - En scalaire et en SIMD (SSE)
  - En mono-thread et en multi-thread (OpenMP)
  - Métrique : cpp (cycle par point de calcul = temps normalisé)

archi	prog	cpp Planar		cpp Halfpipe		gain	temps (ms)
		mono	SMT	mono	SMT		
Dual Penryn T9600 2.8 GHz	scalaire	140.7	81.6	32.0	16.1	29.9	0.440
	SIMD	53.7	51.6	8.4	**4.7**		
bi-Quad Yorkfield	scalaire	**145.0**	16.9	32.0	4.3	90.6	0.140
	SIMD	55.0	3.6	8.4	**1.6**		
bi-quad Nehalem X5550 2.67 GHz	scalaire	**49.0**	7.2	24.5	3.4	35.0	0.110
	SIMD	18.6	3.3	6.0	**1.4**		
Cell	scalaire	**857.0**	402.0	199.0	140.0	214.3	0.328
	SIMD	79.5	12.6	29.8	**4.0**		
GeForce 9400 M	scalaire	27.6		**21.9**		1.3	5.182
GeForce GTX285	scalaire	2.2		**1.5**		1.5	0.260

- Impact majeur des optimisations: x91 pour le bi-Yorkfield
- Multi-cœurs SIMD plus efficaces que Cell et GPU (actuellement)

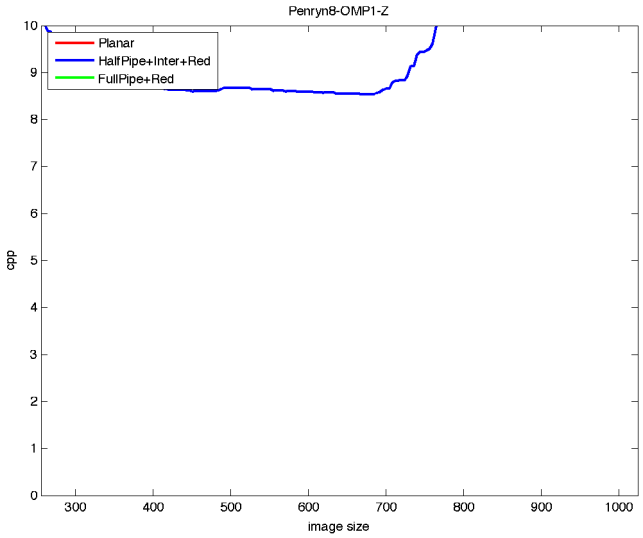
Intérêt des multi-cœurs & Amdhal

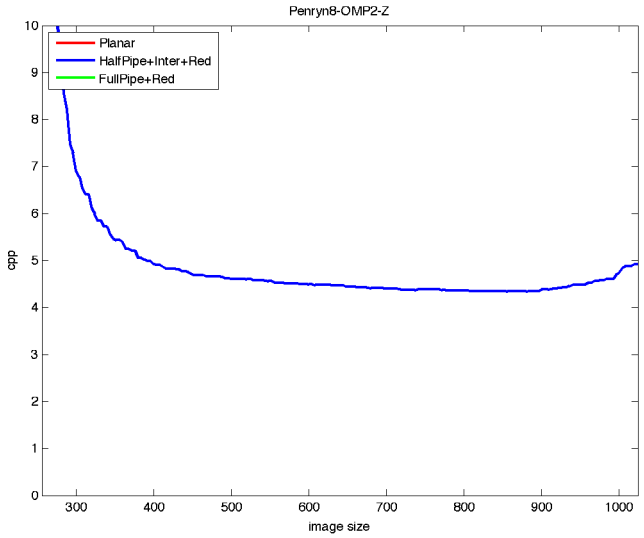
- Origine du gain sur linéaire: la taille du cache
- Augmenter la taille du cache = améliorer les performances (si les données tiennent dans le cache)
- Figures: cpp vs taille des données

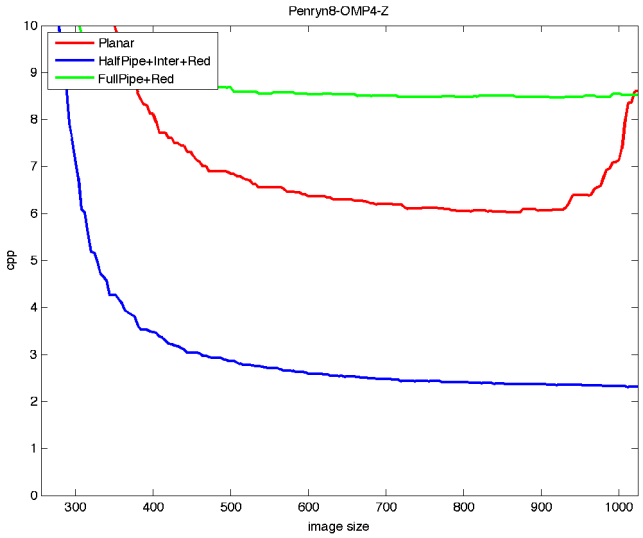


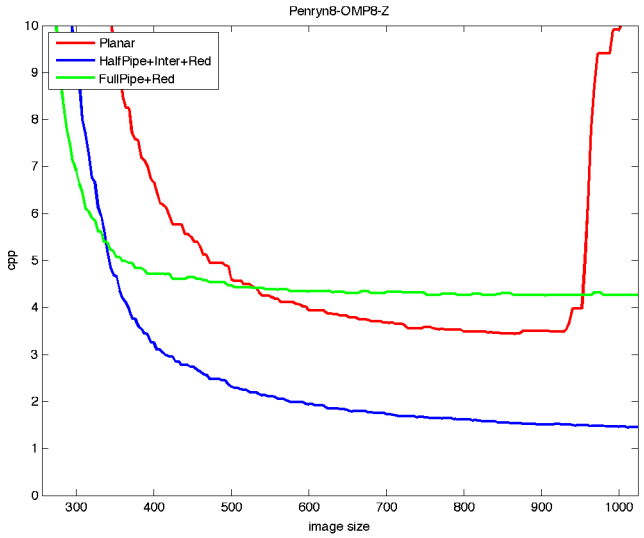
Planar vs Halfpipe : bi-Yorkfield

- 1, 2, 4, 8 cœurs [256 : 1024]



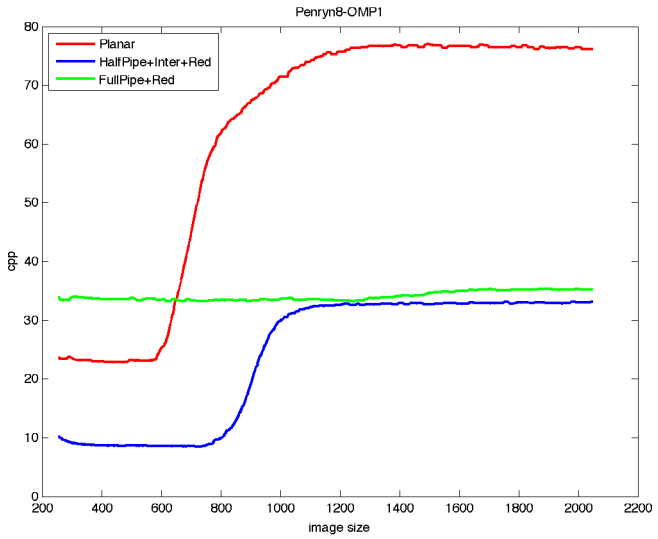


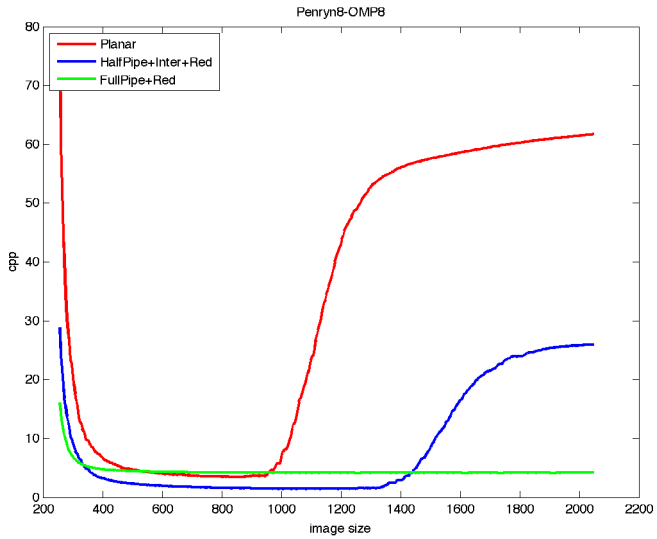




Halfpipe vs Fullpipe : bi-Yorfield

- 1 et 8 cœurs [256 : 2048]





- Fullpipe devient plus beaucoup rapide pour les images de grande taille
- Choisir les transformations en fonction de la taille des données

- Résultats similaire avec buffers ciculaires pour diminuer l'empreinte mémoire
  (très complexe à implanter en multi-thread)

Explications GPU

- CUDA Zone
  - problème du *fairplay* des benchmarks publiés
    transfert non pris en compte dans performance
    comparaison à code mono-thread scalaire …

- Raisons de la contre performance
  - Convolutions nécessitent prise en compte des bords (de l'image, mais aussi des tuiles lors du découpage)
  - mémoire de texture (+lent, mais adressage modulaire / avec blocage) = copie
  - Bus PCIe 16x (à partager si multi-GPU 😐)

  - Temps de transfert >> temps de calcul (GTX285: x8)
    (PCIe v2 = x2, mais NVIDIA Fermi = x2)

  - Rapide ne veut pas dire efficace (ratio puissance crête/puissance utile)

  - Calculs efficaces sur GPU = calculs avec intensité arithmétique élevée
  - TI = acquisition caméra + transferts périodiques. Les algorithmes avec beaucoup de contrôle sont difficilement accélérables.

Comparaison énergétique

- Images 512x512
  - GPU: temps de calcul uniquement

Archi	techno (nm)	Puissance (W)	Temps (ms)	Energie (mJ)
Penryn U9300	45	10	2.58	25.8
Penryn P8700	45	25	2.3	**57.5**
Penryn T9600	45	35	0.44	**15.4**
bi-Yorkfield	45	2x95	0.14	26.6
bi-Nehalem	45	2x130	0.11	29.7
Cell	65	70	0.33	**22.9**
GeForce 9400M	55	12	5.12	**62.2**
GeForce GTX285	55	183	0.26	**47.6**

  - Le Cell reste compétitif
  - Les GPU sont distancés: x3 moins efficaces qu'un portable dual cœurs
  - Temps total GPU: 8 fois pire

De la relativité de la mesure …

- Images 300x300

Archi	techno (nm)	Puissance (W)	Temps (ms)	Energie (mJ)
Penryn U9300	45	10	0.36	**3.6**
Penryn P8700	45	25	0.157	3.9
Penryn T9600	45	35	0.148	5.2
bi-Yorkfield	45	2x95	0.048	**9.1**
bi-Nehalem	45	2x130	0.038	**9.8**
Cell	65	70	0.113	**7.9**

– Importance de la taille des données vis-à-vis de la taille des caches

Conclusion

- Multi-cœurs, Cell ou GPU ?
  - Bouleversement de l'ordre a-priori des performances (puissance crête vs puissance utile)
  - Impact majeur des optimisations
  - Grande variabilité des gains par classes d'algorithmes, mais aussi à cause des détails (ex: imagerie médicale & Monte-Carlo)

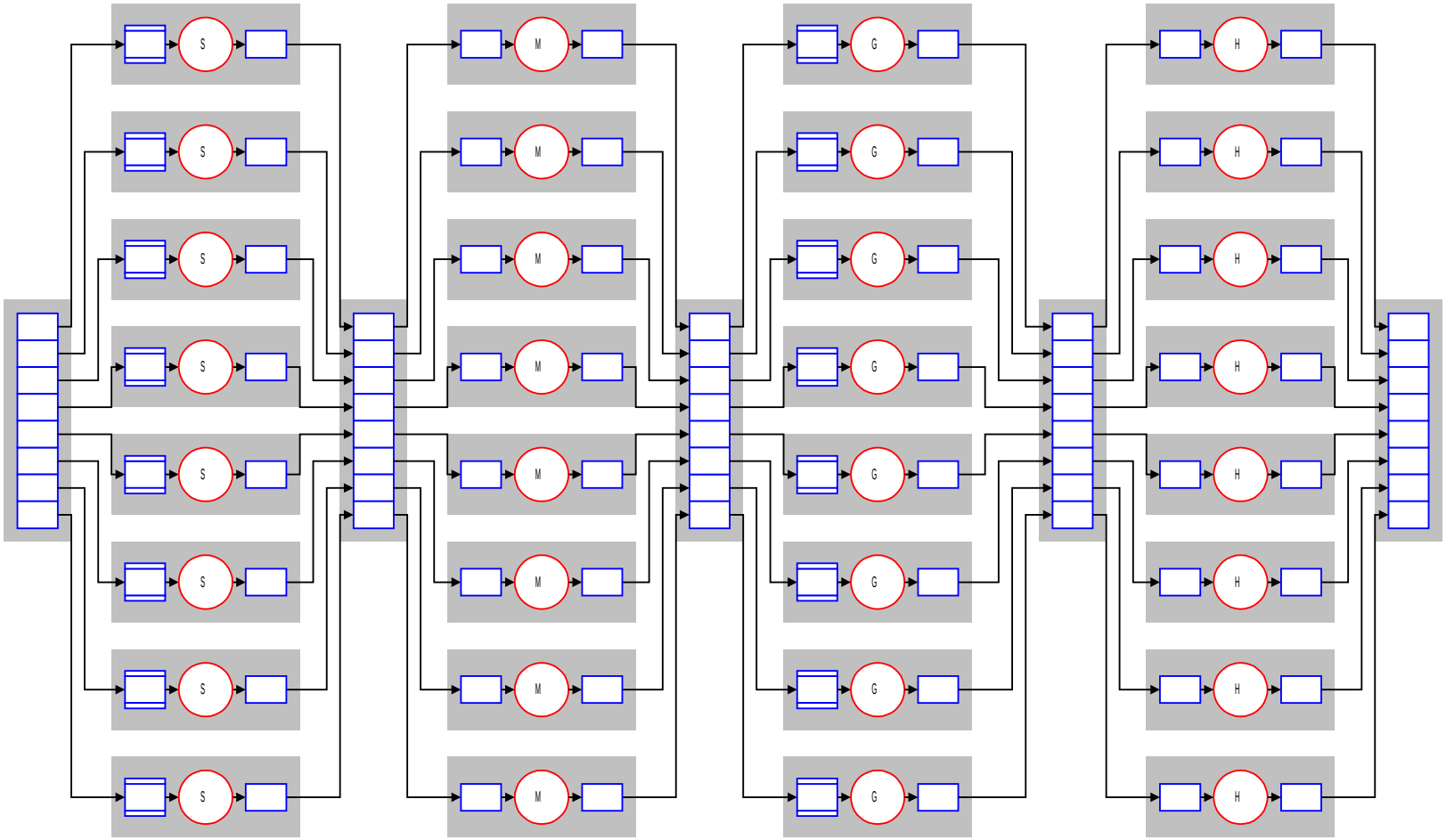
- Faut il recoder pour avoir le maximum de performance ?
  - Oui en général
  - Quid des gros codes ?
    -> quid des architectures futures encore plus complexes ?
    (architectures manycore ex: Intel Terascale Polaris-Larrabee-SCC)
  - Problème de masse critique: expert domaine + numéricien + architecte + informaticien + outils + ...

- Les outils commerciaux (condensation de l'expertise dans des outils)
  - CAPS Entreprise: HMPP
  - HPC-Project: PIPS et Par4All

- Les outils universitaires: en cours ...

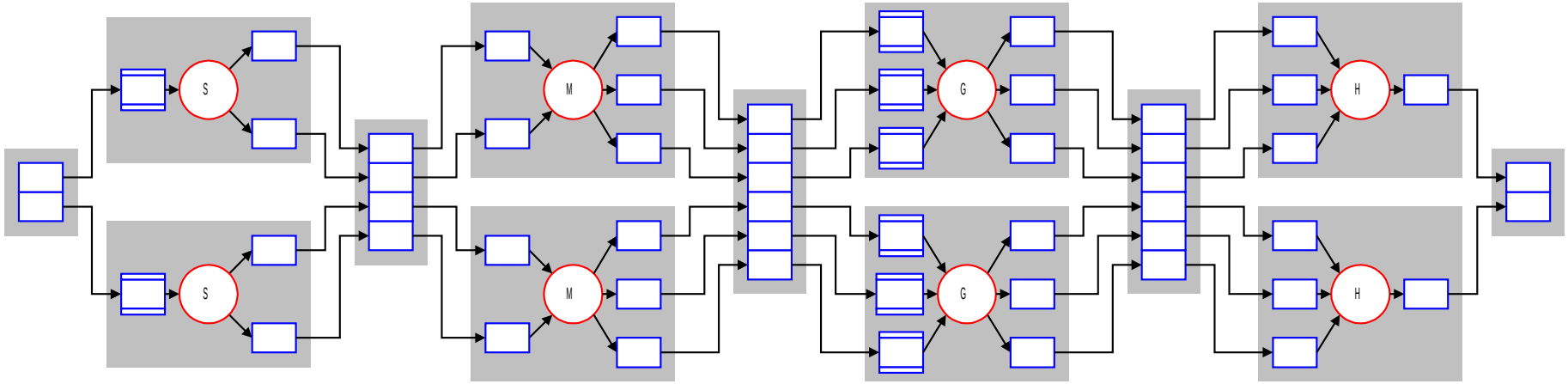
Merci

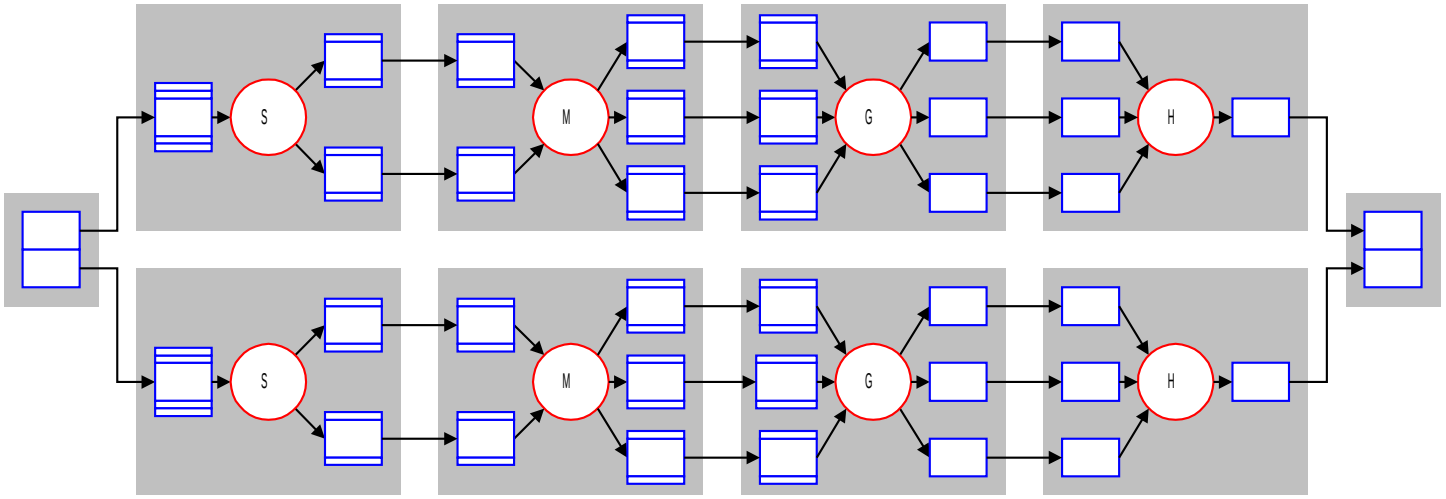
Cell: SPMD classique



Cell: pipeline classique

- Avec/sans accès à la mémoire externe





Cell: nopipe et halfpipe