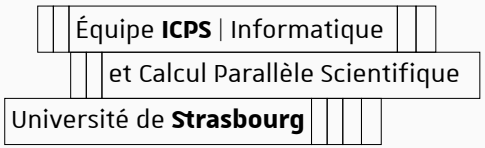
Nids de boucles polyédriques et schedules OpenMP

30 janvier 2019, Journées Compilation, Dammarie-les-Lys

Harenome Ranaivoarivony-Razanajato, Vincent Loechner, Cédric Bastoul

Université de Strasbourg et Inria



Motivation

```
1   #pragma omp parallel for
2   for (int i = 0; i < N; ++i)
3     for (int j = 0; j < i; ++j)
4       C[i] += A[j] * B[j];
```

Motivation

```
1  #pragma omp parallel for
2  for (int i = 0; i < N; ++i)
3    for (int j = 0; j < i; ++j)
4      C[i] += A[j] * B[j];
```

```
1  #pragma omp parallel for schedule(dynamic, 1)
2  for (int i = 0; i < N; ++i)
3    for (int j = 0; j < i; ++j)
4      C[i] += A[j] * B[j];
```

Motivation

```
1   #pragma omp parallel for
2   for (int i = 0; i < N; ++i)
3     for (int j = 0; j < i; ++j)
4       C[i] += A[j] * B[j];
```

```
1   #pragma omp parallel for schedule(dynamic, 1)
2   for (int i = 0; i < N; ++i)
3     for (int j = 0; j < i; ++j)
4       C[i] += A[j] * B[j];
```

	omp parallel for	omp parallel for schedule(dynamic, 1)
Temps d'éxécution	.947692*s*	.55357*s*
Speedup	1.0	1.711

Buts

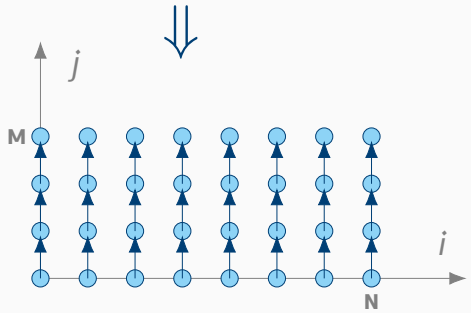
Ojectifs :

- Diminuer le temps d'inactivité des threads en explorant les possibilités offertes par la clause `schedule`
- Déterminer les caractéristiques de nids de boucles polyédriques adéquates pour l'utilisation de cette clause

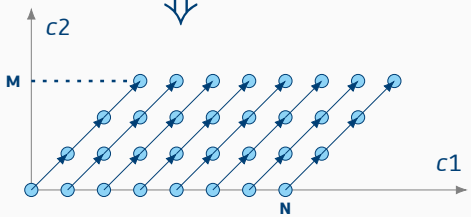
Le modèle polyédrique

```
1   for (i = 0; i <= N; ++i)
2     for (j = 0; j <= M; ++j)
3       S(i, j);
```

1 Modélisation



2 Transformation



3 Génération de code

```
1   for (c1 = 0; c1 <= N+M; c1++)
2     for (c2 = max(0, c1-N); c2<= min(M, c1); c2++)
3       S(c1, c2);
```

Rappels sur OpenMP

- Directives `#pragma omp` pour le calcul parallèle sur architecture à mémoire partagée.
- Parallélisme au sein de régions parallèles (`#pragma omp parallel`)
- Boucles parallèles (`#pragma omp for` ou `#pragma omp parallel for`) : itérations distribuées par *chunk* aux *threads*.

Schedules OpenMP

Arguments de la clause `schedule` pour les boucles `#pragma omp for`

- modifier (`monotonic`, `nonmonotonic`, `simd`)
- **`kind`** (**`static`**, **`dynamic`**, **`guided`**, `auto`, `runtime`)
- **`chunk_size`**

Schedules OpenMP

- **`static`** : les itérations sont distribuées par *chunks* de taille `chunk_size` (sauf le dernier *chunk*) entre les threads en round-robin. Si `chunk_size` n'est pas spécifié, chaque thread reçoit, au plus, un *chunk*.
- **`dynamic`** : les threads demandent un nouveau *chunk* d'itérations lorsqu'ils ont terminé un bloc. La taille des blocs d'itérations est constante (sauf pour le dernier *chunk*) et vaut `1` par défaut si elle n'est pas précisée.
- **`guided`** : les threads demandent des *chunk* d'itérations. La taille des blocs peut changer mais est toujours supérieure à `chunk_size` (sauf pour le dernier *chunk*).

Intuitions

Intuitions : nids de boucles hyper-rectangulaire

Nids de boucles hyper-rectangulaire : chaque contrainte du domaine d'itération est une combinaison linéaire qui implique au plus un itérateur.

```
1  for (int i = 0; i < N; ++i)
2    for (int j = 0; j < M; ++j)
3      S1(i, j);
```

$$\mathcal{D}_{S1}\begin{pmatrix} N \\ M \end{pmatrix} = \left\{ () \to \begin{pmatrix} i \\ j \end{pmatrix} \middle| \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \\ N \\ M \\ 1 \end{pmatrix} \vec{0} \right\}$$

Figure 2: Exemple de nid de boucles hyper-rectangulaire

$\longrightarrow$ *schedule* `guided` ou `static`

Intuitions : nids de boucles non hyper-rectangulaires

Nids de boucles non hyper-rectangulaire : au moins une contrainte du domaine d'itération est une combinaison linéaire de deux itérateurs ou plus.

```
1  for (int i = 0; i < N; ++i)
2    for (int j = i; j < M; ++j)
3      S2(i, j);
```

$$\mathcal{D}_{S2}\begin{pmatrix} N \\ M \end{pmatrix} = \left\{ () \to \begin{pmatrix} i \\ j \end{pmatrix} \middle| \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \\ N \\ M \\ 1 \end{pmatrix} \vec{0} \right\}$$

Figure 3: Exemple de nid de boucles non hyper-rectangulaire

$\longrightarrow$ *schedule* `dynamic`

Intuitions : `chunk_size`

- But : réduire l'inactivité des threads à l'approche du dernier *chunk*
- Pas de dernier *chunk* de plus petite taille ?

  $\longrightarrow$ diviseur du nombre d'itérations total
- Sinon : tenter de minimiser la taille du dernier *chunk* ?

Résultats expérimentaux

Conditions expérimentales

- Suite de benchmarks : PolyBench [2] 4.1
- Code généré par PLUTO [1] 0.11.4 avec les options :
  - `--parallel`
  - `--tile --parallel`
- Combinaisons des *kind* : `dynamic`, `guided`, `static`
  et `chunk_size` : 1, 8, 16, 32
- Code compilé avec `gcc 8.2.1` avec les options `-O3 -march=native -fopenmp`
- Testé sur Intel Xeon E5-2620v3 @ 2.40GHz, linux 4.20
- Temps mesurés avec les scripts fournis dans PolyBench

Meilleures versions

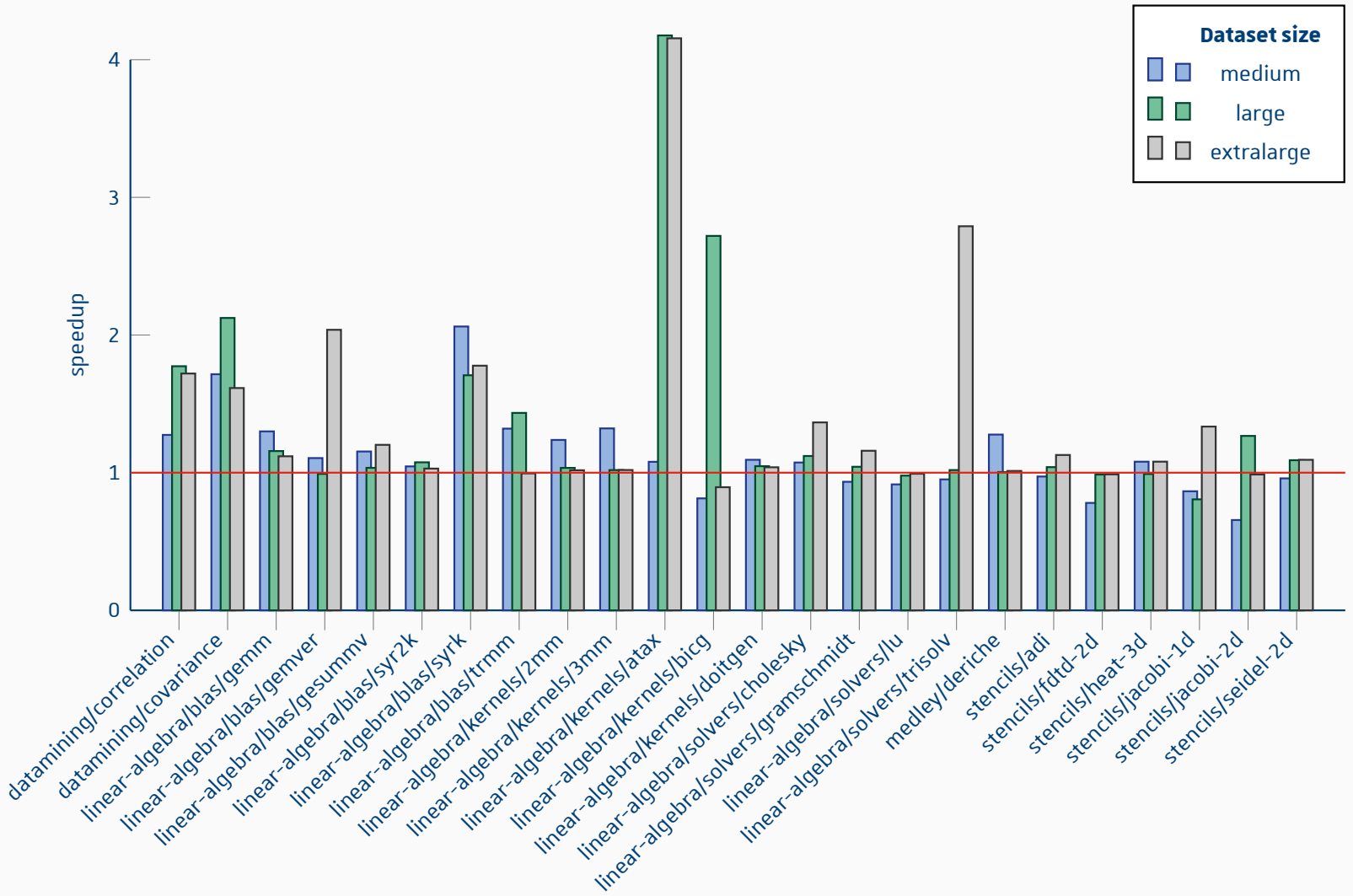


Figure 4: Speedup par rapport à PLUTO, Intel Xeon E5-2620v3 (6 coeurs / 12threads)

Caractéristiques des meilleurs codes

benchmark	rectangulaire	medium	N	large	N	extralarge	N
datamining/**correlation**	non	(dynamic, 8)	240	(dynamic, 8)	1200	(dynamic, 8)	2600
datamining/**covariance**	non	(dynamic, 1)	240	(dynamic, 8)	1200	(dynamic, 32)	2600
linear-algebra/blas/**gemm**	oui	(guided, 1)	200	(guided, 8)	1000	(dynamic, 1)	2000
linear-algebra/blas/**gemver**	oui	(guided, 16)	400	(static, 32)	2000	(static, 8)	4000
linear-algebra/blas/**gesummv**	oui	(dynamic, 8)	250	(guided, 8)	1300	(dynamic, 8)	2800
linear-algebra/blas/**syr2k**	oui	(dynamic, 1)	200	(dynamic, 1)	1000	(guided, 1)	2000
linear-algebra/blas/**syrk**	non	(dynamic, 1)	200	(dynamic, 1)	1000	(dynamic, 1)	2000
linear-algebra/blas/**trmm**	oui	(static, 8)	200	(guided, 1)	1000	(static, 16)	2000
linear-algebra/kernels/**2mm**	oui	(dynamic, 1)	180	(dynamic, 1)	800	(dynamic, 8)	1600
linear-algebra/kernels/**3mm**	oui	(guided, 1)	180	(guided, 1)	800	(guided, 1)	1600
linear-algebra/kernels/**atax**	non	(static, 32)	390	(static, 32)	1900	(static, 1)	1800
linear-algebra/kernels/**bicg**	non	(static, 8)	390	(static, 32)	1900	(static, 16)	1800
linear-algebra/kernels/**doitgen**	oui	(static, 8)	40	(static, 32)	140	(static, 32)	220
linear-algebra/solvers/**cholesky**	non	(dynamic, 8)	400	(guided, 1)	2000	(guided, 8)	4000
linear-algebra/solvers/**gramschmidt**	oui	(guided, 8)	200	(dynamic, 1)	400	(dynamic, 1)	2000
linear-algebra/solvers/**lu**	non	(static, 32)	400	(guided, 32)	1000	(static, 32)	2000
linear-algebra/solvers/**trisolv**	oui	(static, 32)	400	(static, 32)	2000	(static, 32)	4000
medley/**deriche**	oui	(static, 32)	720	(guided, 1)	4096	(guided, 32)	7680
stencils/**adi**	oui	(guided, 1)	200	(guided, 1)	1000	(dynamic, 1)	2000
stencils/**fdtd-2d**	non	(guided, 1)	200	(guided, 1)	1000	(guided, 32)	2000
stencils/**heat-3d**	non	(dynamic, 1)	40	(dynamic, 8)	120	(guided, 8)	200
stencils/**jacobi-1d**	non	(dynamic, 16)	400	(dynamic, 32)	2000	(dynamic, 16)	4000
stencils/**jacobi-2d**	non	(dynamic, 1)	250	(guided, 1)	1300	(dynamic, 32)	2800
stencils/**seidel-2d**	non	(dynamic, 1)	400	(dynamic, 1)	2000	(dynamic, 1)	4000

Pires versions

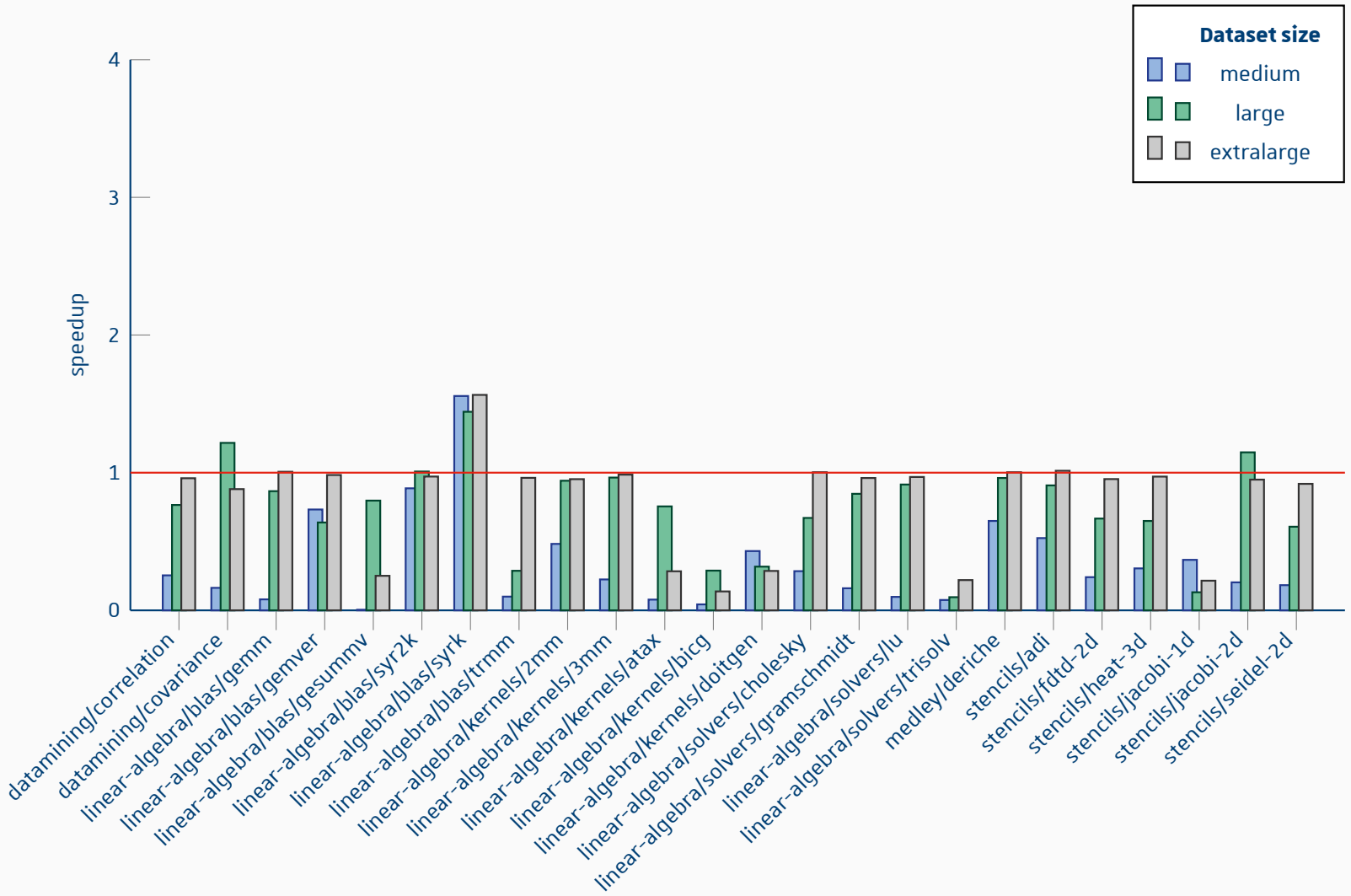


Figure 5: Slowdown par rapport à PLUTO, Intel Xeon E5-2620v3 (6 coeurs / 12threads)

Comparaison meilleures / pires versions

benchmark	medium		large		extralarge	
datamining/**correlation**	(dynamic, 8)	(static, 32)	(dynamic, 8)	(guided, 32)	(dynamic, 8)	(guided, 1)
datamining/**covariance**	(dynamic, 1)	(static, 8)	(dynamic, 8)	(static, 16)	(dynamic, 32)	(guided, 8)
linear-algebra/blas/**gemm**	(guided, 1)	(static, 16)	(guided, 8)	(static, 1)	(dynamic, 1)	(static, 1)
linear-algebra/blas/**gemver**	(guided, 16)	(dynamic, 8)	(static, 32)	(dynamic, 32)	(static, 8)	(guided, 16)
linear-algebra/blas/**gesummv**	(dynamic, 8)	(dynamic, 16)	(guided, 8)	(static, 32)	(dynamic, 8)	(dynamic, 32)
linear-algebra/blas/**syr2k**	(dynamic, 1)	(static, 1)	(dynamic, 1)	(static, 1)	(guided, 1)	(static, 1)
linear-algebra/blas/**syrk**	(dynamic, 1)	(dynamic, 16)	(dynamic, 1)	(dynamic, 32)	(dynamic, 1)	(dynamic, 1)
linear-algebra/blas/**trmm**	(static, 8)	(static, 1)	(guided, 1)	(dynamic, 16)	(static, 16)	(dynamic, 8)
linear-algebra/kernels/**2mm**	(dynamic, 1)	(static, 32)	(dynamic, 1)	(dynamic, 32)	(dynamic, 8)	(static, 32)
linear-algebra/kernels/**3mm**	(guided, 1)	(static, 32)	(guided, 1)	(static, 32)	(guided, 1)	(dynamic, 32)
linear-algebra/kernels/**atax**	(static, 32)	(guided, 32)	(static, 32)	(guided, 16)	(static, 1)	(dynamic, 1)
linear-algebra/kernels/**bicg**	(static, 8)	(guided, 1)	(static, 32)	(dynamic, 1)	(static, 16)	(guided, 1)
linear-algebra/kernels/**doitgen**	(static, 8)	(guided, 1)	(static, 32)	(guided, 1)	(static, 32)	(guided, 1)
linear-algebra/solvers/**cholesky**	(dynamic, 8)	(guided, 8)	(guided, 1)	(static, 32)	(guided, 8)	(dynamic, 32)
linear-algebra/solvers/**gramschmidt**	(guided, 8)	(static, 1)	(dynamic, 1)	(dynamic, 32)	(dynamic, 1)	(static, 32)
linear-algebra/solvers/**lu**	(static, 32)	(dynamic, 32)	(guided, 32)	(dynamic, 32)	(static, 32)	(dynamic, 16)
linear-algebra/solvers/**trisolv**	(static, 32)	(guided, 16)	(static, 32)	(dynamic, 1)	(static, 32)	(dynamic, 1)
medley/**deriche**	(static, 32)	(static, 8)	(guided, 1)	(dynamic, 16)	(guided, 32)	(guided, 8)
stencils/**adi**	(guided, 1)	(guided, 32)	(guided, 1)	(dynamic, 32)	(dynamic, 1)	(dynamic, 32)
stencils/**fdtd-2d**	(guided, 1)	(dynamic, 32)	(guided, 1)	(dynamic, 16)	(guided, 32)	(dynamic, 1)
stencils/**heat-3d**	(dynamic, 1)	(dynamic, 16)	(dynamic, 8)	(dynamic, 32)	(guided, 8)	(dynamic, 1)
stencils/**jacobi-1d**	(dynamic, 16)	(guided, 1)	(dynamic, 32)	(dynamic, 1)	(dynamic, 16)	(dynamic, 1)
stencils/**jacobi-2d**	(dynamic, 1)	(dynamic, 32)	(guided, 1)	(guided, 32)	(dynamic, 32)	(guided, 1)
stencils/**seidel-2d**	(dynamic, 1)	(guided, 32)	(dynamic, 1)	(dynamic, 32)	(dynamic, 1)	(guided, 32)

Meilleurs versions : nids de boucles tuilées

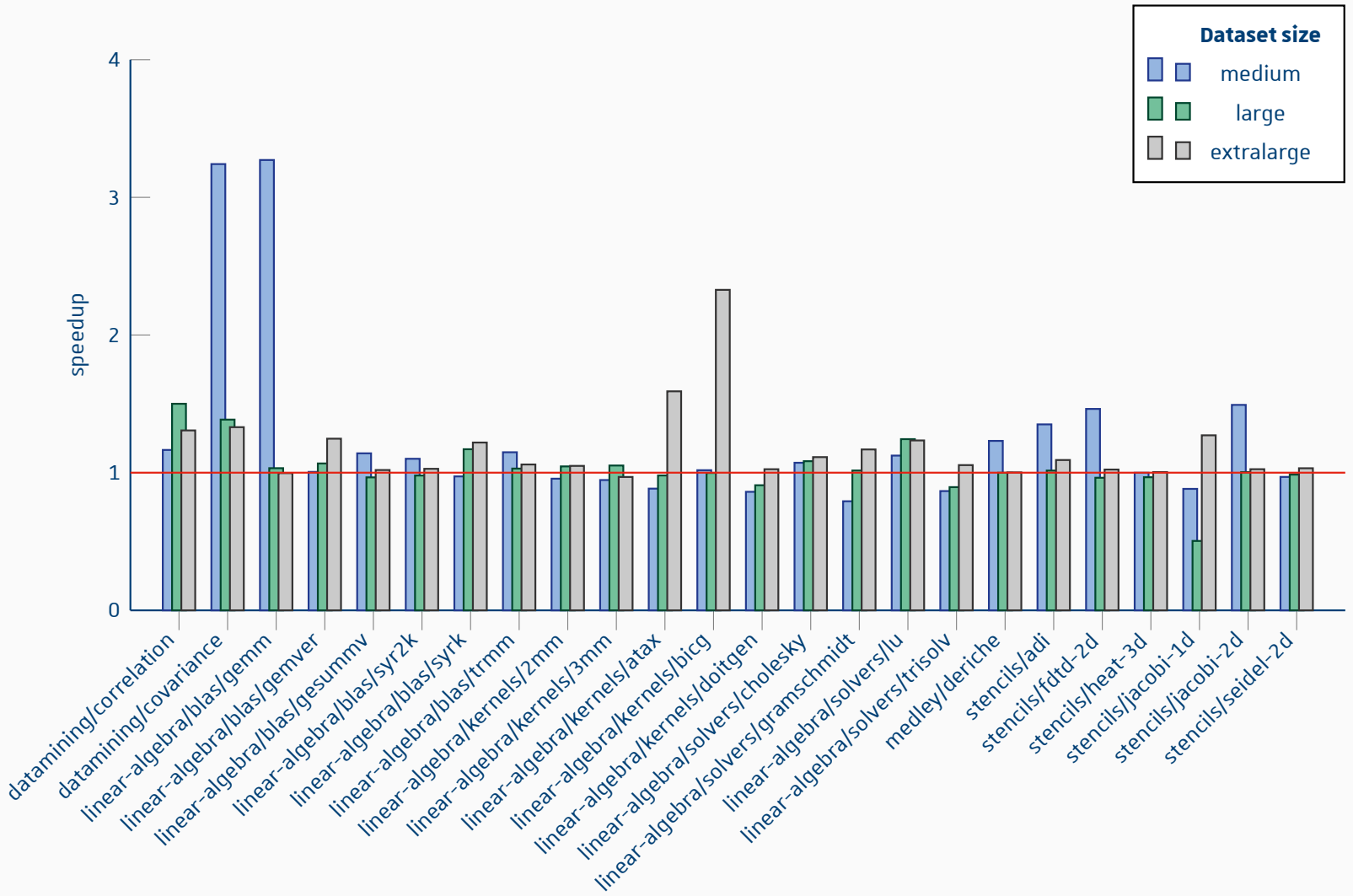


Figure 6: Speedup par rapport à PLUTO, Intel Xeon E5-2620v3 (6 coeurs / 12threads)

Meilleurs codes : nids de boucles tuilées

benchmark	medium	large	extralarge
datamining/**correlation**	static, 1	static, 1	dynamic, 1
datamining/**covariance**	dynamic, 1	dynamic, 1	dynamic, 1
linear-algebra/blas/**gemm**	static, 1	static, 1	dynamic, 1
linear-algebra/blas/**gemver**	static, 1	guided, 1	dynamic, 1
linear-algebra/blas/**gesummv**	guided, 1	dynamic, 1	static, 8
linear-algebra/blas/**syr2k**	dynamic, 1	static, 1	dynamic, 1
linear-algebra/blas/**syrk**	static, 1	dynamic, 1	dynamic, 1
linear-algebra/blas/**trmm**	static, 1	static, 1	dynamic, 1
linear-algebra/kernels/**2mm**	static, 1	guided, 1	guided, 1
linear-algebra/kernels/**3mm**	static, 1	static, 1	static, 1
linear-algebra/kernels/**atax**	static, 1	static, 1	guided, 1
linear-algebra/kernels/**bicg**	guided, 1	guided, 1	guided, 8
linear-algebra/kernels/**doitgen**	static, 32	dynamic, 1	static, 1
linear-algebra/solvers/**cholesky**	static, 1	guided, 1	dynamic, 1
linear-algebra/solvers/**gramschmidt**	static, 1	guided, 1	guided, 1
linear-algebra/solvers/**lu**	guided, 1	dynamic, 1	dynamic, 1
linear-algebra/solvers/**trisolv**	dynamic, 1	static, 1	dynamic, 1
medley/**deriche**	dynamic, 8	static, 1	guided, 1
stencils/**adi**	dynamic, 1	dynamic, 1	dynamic, 1
stencils/**fdtd-2d**	guided, 1	static, 1	dynamic, 1
stencils/**heat-3d**	dynamic, 1	static, 1	static, 1
stencils/**jacobi-1d**	static, 1	static, 1	static, 1
stencils/**jacobi-2d**	guided, 1	static, 1	dynamic, 1
stencils/**seidel-2d**	static, 1	static, 1	dynamic, 1

Conclusion et perspectives

Conclusion et perspectives

- Observations :
  - Définitivement un intérêt à spécifier `schedule` et `chunk_size`
  - Domaines tuilés : `chunk_size` = 1 et `dynamic` semblent les plus appropriés.
  - Domaines non tuilés non hyper-rectangulaires : `schedule(dynamic)` généralement le plus intéréssant
  - Petites valeurs de `chunk_size` pour les schedules `dynamic` et `guided`
  - Grandes valeurs de `chunk_size` pour les schedules `static`
  - Pas évident de trouver le bon couple `kind`/`chunk_size`
- Futurs travaux :
  - Méthode pour déterminer `chunk_size`
  - Interaction avec la clause `nowait` [3] lors de l'utilisation de `schedule(static, <chunk_size>)` dans des régions parallèles

Références i

[1] Uday Bondhugula et al. “A Practical Automatic Polyhedral Parallelizer and Locality Optimizer”. In: *Acm Sigplan Notices*. Vol. 43. 6. ACM. 2008, pp. 101–113.

[2] Louis-Noël Pouchet. “Polybench: The polyhedral benchmark suite”. In: *URL: http://www. cs. ucla. edu/pouchet/software/polybench* (2012).

[3] Harenome Razanajato, Cédric Bastoul, and Vincent Loechner. “Lifting Barriers Using Parallel Polyhedral Regions”. In: *High Performance Computing (HiPC), 2017 IEEE 24th International Conference on*. IEEE. 2017, pp. 338–347.