富士通詳細プロファイラの利用事例

2022-12-14

17th HPC-Phys勉強会

R-CCSソフトウェア開発技術ユニット

似鳥啓吾

内容

- 詳細プロファイラfappの使い方
  - 雰囲気で使っているので非公式です、あくまでも１ユーザーの体験と感想として
- Nek5000/RS axhelmカーネルのチューニング
  - R-CCS辻さんの仕事の再現実験＋文書化
  - プロファイラを取ったのはチューニングの後
- Bridge++ QXS Wilsonカーネルのチューニング
  - 金森さんからの宿題
  - チューニング前にプロファイラを取った例

参考文献

- 「富岳」もしくはFX1000ユーザー限定となります
- ポータル->マニュアル->言語マニュアル
  - プロファイラ使用手引書
- ポータル->プログラミングガイド
  - プログラミングガイド (チューニング編)
  - プログラミングガイド (プロセッサ編)

基本プロファイラと詳細プロファイラ

- 基本プロファイラ: fipp: Fujitsu Instant Performance Profiler
  - 割り込みによるサンプリングベース
  - 関数ごとの消費時間など
  - 一部CPU動作状況（Gflopsやメモリスループット）
- 詳細プロファイラ: fapp: Fujitsu Advanced Performance Profiler
  - ハードウェアカウンターベース
  - 最大17回同じプログラムを繰り返し実行
  - CSV形式に変換後Excelマクロで可視化できる

- 共通事項
  - 実行時のDLL切り替えにより通常実行とプロファイリングを切り替え
  - ソースコードにstart/stopの埋め込みが必要だが、プロファイリング専用のbranchを作らなくても済む
  - ただし富士通以外の環境ともソースを共通化するにはマクロ関数でAPIを消せるようにする必要がある

ソースコードの編集 (fapp)

```
#include <fj_tool/fapp.h>

 fapp_start("eo_bulk", 1, 0);
 BridgeQXS::mult_wilson_eo_bulk_dirac(v2, up, v1, xp, m_CKs, &m_boundary[0],
                    m_Nsize, do_comm, &m_Leo[0], ieo, iflag);
 fapp_stop("eo_bulk", 1, 0);
```

- name, number, levelの順で指定
  - name+numberで区間を識別
  - levelは0なら常に計測、実行時にオプションで対象を絞れる
- fapp_start/fapp_stopのネストも可能

実行スクリプトの例

```
for i in {1..17}
do
  fapp -C -d ./rep${i} -Hevent=pa${i},method=fast mpiexec ${BIN} ${Px} ${Py} ${Pz} ${Pt} ${Nx} ${Ny} ${Nz}
${Nt} ${Nthreads} ${Iter}
done

for i in {1..17}
do
  fapp -A -d ./rep${i} -Icpupa,nompi -tcsv -o pa${i}.csv
done
```

- ホストでcsvファイルを作成するときはfapppxコマンド
- -Hmethod=fastだとOSを経由しないのでオーバーヘッドが小さい

- 後々結果をscpする用に/optからxlsmファイルへのリンクを作っておくとよい

CPU Performance Analysis Report 4.2.1

Measured time	Wed Nov 16 22:18:10 2022
Node name	e01-0100c

Process no.	0
CMG no.	0
Measured region	eo_bulk, 1

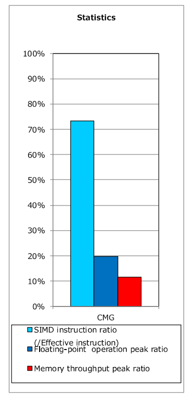
Vector length (bit)	512
CPU frequency (GHz)	1.885

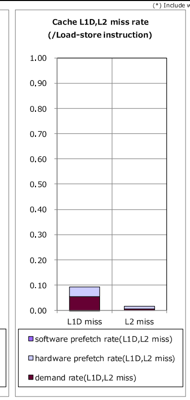
Statistics Process	Thread	Execution time (s)	GFLOPS	Floating-point operation peak ratio (%)	Memory throughput (GB/s)	Memory throughput peak ratio (%)	Effective instruction	Floating-point operation	SIMD instruction rate (%) (/Effective instruction)	SVE operation rate (%)	Floating-point pipeline Active element rate (%)	IPC	GIPS
0	0	1.14E-01	10.78	17.87%	3.04	11.52%	1.73E+08	1.23E+09	72.58%	100.00%	73.77%	0.80	1.52
0	1	1.17E-01	11.36	18.84%	2.68		1.82E+08	1.33E+09	72.94%	100.00%	74.56%	0.81	1.56
0	2	1.18E-01	12.45	20.63%	2.40		1.94E+08	1.47E+09	73.53%	100.00%	75.57%	0.85	1.64
0	3	1.14E-01	12.59	20.87%	2.51		1.89E+08	1.44E+09	73.44%	100.00%	75.62%	0.86	1.65
0	4	1.18E-01	12.25	20.31%	2.22		1.91E+08	1.44E+09	73.36%	100.00%	75.49%	0.85	1.62
0	5	1.21E-01	12.14	20.13%	2.83		1.94E+08	1.47E+09	73.53%	100.00%	75.56%	0.83	1.60
0	6	1.19E-01	12.05	19.98%	2.48		1.89E+08	1.44E+09	73.45%	100.00%	75.60%	0.83	1.58
0	7	1.22E-01	11.82	19.59%	2.25		1.91E+08	1.44E+09	73.37%	100.00%	75.46%	0.82	1.56
0	8	1.24E-01	11.86	19.66%	2.37		1.94E+08	1.47E+09	73.53%	100.00%	75.58%	0.81	1.56
0	9	1.23E-01	11.67	19.34%	2.65		1.89E+08	1.44E+09	73.44%	100.00%	75.58%	0.80	1.53
0	10	1.14E-01	11.73	19.44%	1.85		1.83E+08	1.34E+09	73.04%	100.00%	74.60%	0.84	1.60
0	11	1.05E-01	12.03	19.94%	2.21		1.77E+08	1.26E+09	72.88%	100.00%	73.84%	0.89	1.69
CMG 0 total		1.18E-01	142.74	19.72%	29.50	11.52%	2.24E+09	1.68E+10	73.27%	100.00%	75.13%	0.83	19.09

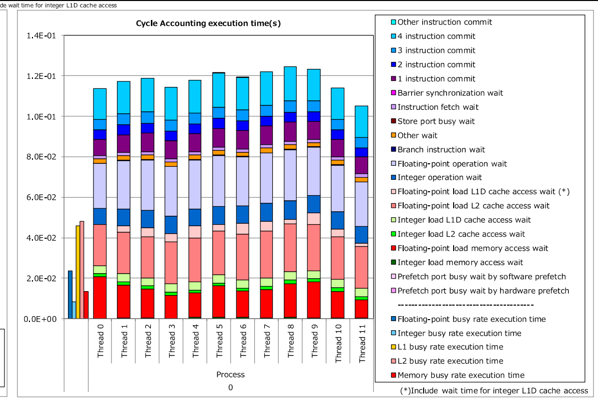
Cycle Accounting Process	Thread	Prefetch port busy wait: Prefetch port busy wait by hardware prefetch	Prefetch port busy wait by software prefetch	Memory access wait & Cache access wait: Integer load memory access wait	Floating-point load memory access wait	Integer load L2 cache access wait	Integer load L1D cache access wait	Floating-point load L2 cache access wait	Floating-point load L1D cache access wait (*)	Operation wait: Integer operation wait	Floating-point operation wait	Other wait: Branch instruction wait	Other wait	Store port busy wait	Instruction fetch wait	Barrier synchronization wait	1 instruction commit	Other instruction commit: 2 instruction commit	3 instruction commit	4 instruction commit	Other instruction commit	Total
0	0	0.00E+00	0.00E+00	2.90E-05	2.07E-02	1.51E-03	3.92E-03	2.02E-02	0.00E+00	8.00E-03	2.22E-02	1.66E-04	2.16E-03	0.00E+00	1.52E-03	0.00E+00	8.03E-03	4.72E-03	5.11E-03	1.55E-02	0.00E+00	1.14E-01
0	1	0.00E+00	0.00E+00	2.83E-04	1.64E-02	1.39E-03	4.22E-03	2.05E-02	3.17E-03	8.29E-03	2.39E-02	1.99E-04	2.25E-03	0.00E+00	1.69E-03	0.00E+00	8.66E-03	4.89E-03	5.34E-03	1.62E-02	0.00E+00	1.17E-01
0	2	0.00E+00	0.00E+00	4.85E-04	1.44E-02	1.33E-03	4.00E-03	2.05E-02	4.25E-03	8.71E-03	2.49E-02	1.89E-04	2.26E-03	0.00E+00	1.58E-03	0.00E+00	9.22E-03	4.84E-03	5.56E-03	1.66E-02	0.00E+00	1.18E-01
0	3	0.00E+00	0.00E+00	6.84E-05	1.16E-02	1.59E-03	3.87E-03	2.09E-02	4.09E-03	8.53E-03	2.44E-02	1.81E-04	2.14E-03	0.00E+00	1.69E-03	0.00E+00	8.96E-03	4.77E-03	5.40E-03	1.62E-02	0.00E+00	1.14E-01
0	4	0.00E+00	0.00E+00	5.78E-04	1.21E-02	1.45E-03	4.17E-03	2.16E-02	5.88E-03	8.30E-03	2.44E-02	1.89E-04	2.22E-03	0.00E+00	1.56E-03	0.00E+00	8.89E-03	4.79E-03	5.39E-03	1.63E-02	0.00E+00	1.18E-01
0	5	0.00E+00	0.00E+00	6.24E-04	1.56E-02	1.42E-03	4.15E-03	2.17E-02	3.20E-03	8.80E-03	2.52E-02	1.90E-04	2.27E-03	0.00E+00	1.60E-03	0.00E+00	9.36E-03	4.91E-03	5.61E-03	1.69E-02	2.60E-05	1.21E-01
0	6	0.00E+00	0.00E+00	8.42E-04	1.28E-02	1.50E-03	4.19E-03	2.23E-02	5.41E-03	8.57E-03	2.43E-02	1.79E-04	2.19E-03	0.00E+00	1.54E-03	0.00E+00	9.04E-03	4.74E-03	5.40E-03	1.61E-02	1.60E-05	1.19E-01
0	7	0.00E+00	0.00E+00	4.53E-04	1.40E-02	1.63E-03	4.13E-03	2.31E-02	4.70E-03	8.93E-03	2.48E-02	1.92E-04	2.29E-03	0.00E+00	1.64E-03	0.00E+00	9.20E-03	4.89E-03	5.48E-03	1.66E-02	0.00E+00	1.22E-01
0	8	0.00E+00	0.00E+00	6.27E-04	1.68E-02	1.86E-03	4.12E-03	2.34E-02	2.17E-03	9.19E-03	2.54E-02	1.89E-04	2.31E-03	0.00E+00	1.60E-03	0.00E+00	9.44E-03	4.92E-03	5.66E-03	1.68E-02	0.00E+00	1.24E-01
0	9	0.00E+00	0.00E+00	2.53E-04	1.78E-02	1.65E-03	4.01E-03	2.28E-02	5.76E-03	8.51E-03	2.40E-02	1.76E-04	2.19E-03	0.00E+00	1.54E-03	0.00E+00	8.94E-03	4.66E-03	5.29E-03	1.57E-02	0.00E+00	1.23E-01
0	10	0.00E+00	0.00E+00	3.87E-04	1.30E-02	2.01E-03	4.09E-03	2.11E-02	3.71E-03	8.56E-03	2.30E-02	1.99E-04	2.19E-03	0.00E+00	1.70E-03	0.00E+00	8.56E-03	4.69E-03	5.17E-03	1.58E-02	0.00E+00	1.14E-01
0	11	0.00E+00	0.00E+00	2.91E-04	9.02E-03	1.67E-03	4.02E-03	2.08E-02	1.45E-03	8.22E-03	2.20E-02	1.96E-04	2.18E-03	0.00E+00	1.88E-03	0.00E+00	8.15E-03	4.53E-03	5.15E-03	1.55E-02	0.00E+00	1.05E-01
CMG 0 total		0.00E+00	0.00E+00	4.10E-04	1.45E-02	1.58E-03	4.07E-03	2.16E-02	3.65E-03	8.55E-03	2.40E-02	1.87E-04	2.22E-03	0.00E+00	1.63E-03	0.00E+00	8.87E-03	4.78E-03	5.38E-03	1.62E-02	3.50E-06	1.18E-01

(*) Include wait time for integer L1D cache access

Busy Process	Thread	Floating-point operation pipeline A busy rate (%)	Floating-point operation pipeline B busy rate (%)	Integer operation pipeline A busy rate (%)	Integer operation pipeline B busy rate (%)	L1 busy rate (%)	L2 busy rate (%)	Memory busy rate (%)	Address calculation operation pipeline A busy rate (%)	Address calculation operation pipeline B busy rate (%)	Floating-point pipeline A Active element rate (%)	Floating-point pipeline B Active element rate (%)	L1 pipeline 0 Active element rate (%)	L1 pipeline 1 Active element rate (%)	SFI(Store Fetch Interlock) rate
0	0	20.83%	18.05%	6.36%	7.99%	38.36%	41.07%	11.52%	20.29%	20.11%	58.67%	91.19%	93.24%	92.29%	0.02
0	1	21.22%	18.46%	6.69%	8.13%	38.34%			21.07%	20.94%	59.77%	91.56%	93.15%	92.35%	0.02
0	2	22.07%	19.53%	6.74%	7.99%	40.27%			21.50%	21.53%	60.92%	92.13%	93.39%	92.62%	0.02
0	3	21.81%	19.24%	6.70%	8.07%	39.52%			21.47%	21.43%	60.96%	92.23%	93.47%	92.64%	0.02
0	4	21.95%	19.33%	6.59%	7.89%	39.44%			20.97%	20.94%	60.89%	92.07%	93.26%	92.53%	0.02
0	5	21.80%	19.23%	6.50%	7.68%	39.66%			20.75%	20.73%	60.93%	92.15%	93.36%	92.65%	0.02
0	6	21.04%	18.57%	6.19%	7.51%	38.37%			19.96%	19.90%	60.92%	92.24%	93.47%	92.65%	0.02
0	7	21.08%	18.58%	6.32%	7.63%	38.29%			20.09%	20.06%	60.81%	92.09%	93.21%	92.50%	0.02
0	8	21.26%	18.80%	6.25%	7.42%	38.95%			20.09%	20.05%	60.92%	92.16%	93.29%	92.63%	0.02
0	9	20.53%	18.10%	5.82%	7.06%	37.55%			18.73%	18.66%	60.87%	92.26%	93.39%	92.77%	0.02
0	10	22.06%	19.37%	6.46%	7.76%	39.85%			20.03%	19.92%	59.67%	91.60%	92.98%	92.22%	0.02
0	11	22.61%	19.82%	7.02%	8.37%	40.84%			21.69%	21.53%	58.64%	91.19%	92.87%	91.96%	0.02
CMG 0 total		21.50%	18.91%	6.46%	7.77%	39.09%	41.07%	11.52%	20.52%	20.45%	60.37%	91.92%	93.26%	92.48%	0.02







Cache Process	Thread	L1I miss rate (/Effective instruction)	Load-store instruction	L1D miss	L1D miss rate (/Load-store instruction)	L1D miss demand rate (%) (/L1D miss)	L1D miss hardware prefetch rate (%) (/L1D miss)	L1D miss software prefetch rate (%) (/L1D miss)	L2 miss	L2 miss rate (/Load-store instruction)	L2 miss demand rate (%) (/L2 miss)	L2 miss hardware prefetch rate (%) (/L2 miss)	L2 miss software prefetch rate (%) (/L2 miss)	L1D TLB miss rate (/Load-store instruction)	L2D TLB miss rate (/Load-store instruction)
0	0	0.00	6.77E+07	6.03E+06	0.09	59.46%	40.54%	0.01%	1.02E+06	0.02	40.17%	78.68%	0.00%	0.00105	0.00000
0	1	0.00	7.10E+07	6.35E+06	0.09	60.93%	39.26%	-0.18%	1.02E+06	0.01	47.89%	71.62%	0.00%	0.00092	0.00000
0	2	0.00	7.49E+07	7.17E+06	0.10	60.00%	39.81%	0.18%	8.73E+05	0.01	49.52%	71.98%	0.00%	0.00089	0.00000
0	3	0.00	7.29E+07	6.89E+06	0.09	60.45%	39.55%	0.00%	9.33E+05	0.01	46.11%	70.63%	0.00%	0.00091	0.00000
0	4	0.00	7.37E+07	6.83E+06	0.09	60.35%	39.30%	0.35%	8.93E+05	0.01	54.71%	60.34%	0.00%	0.00090	0.00000
0	5	0.00	7.49E+07	7.14E+06	0.10	59.95%	40.02%	0.03%	1.11E+06	0.01	43.54%	71.20%	0.00%	0.00089	0.00000
0	6	0.00	7.29E+07	6.86E+06	0.09	60.63%	39.58%	-0.22%	9.58E+05	0.01	40.85%	74.31%	0.00%	0.00091	0.00000
0	7	0.00	7.37E+07	6.88E+06	0.09	60.59%	39.35%	0.06%	8.92E+05	0.01	51.39%	64.44%	0.00%	0.00090	0.00000
0	8	0.00	7.49E+07	7.17E+06	0.10	59.37%	40.64%	-0.01%	9.95E+05	0.01	40.90%	74.45%	0.00%	0.00089	0.00000
0	9	0.00	7.29E+07	6.94E+06	0.10	60.51%	39.72%	-0.23%	1.00E+06	0.01	43.65%	73.29%	0.00%	0.00092	0.00000
0	10	0.00	7.10E+07	6.34E+06	0.09	58.58%	41.56%	-0.15%	6.74E+05	0.01	42.57%	78.55%	0.00%	0.00094	0.00000
0	11	0.00	6.95E+07	6.00E+06	0.09	55.64%	44.74%	-0.39%	7.69E+05	0.01	29.74%	87.36%	0.00%	0.00095	0.00000
CMG 0 total		0.00	8.70E+08	8.06E+07	0.09	59.75%	40.29%	-0.04%	1.11E+07	0.01	43.78%	73.54%	0.00%	0.00092	0.00000

Instruction — Load-store instruction

Process	Thread	Load SIMD: Single vector contiguous load instruction	Multiple vector contiguous structure load instruction	Non-contiguous gather load instruction	Broadcast load instruction	Floating-point register fill instruction	Predicate register fill instruction	First-fault load instruction	Non-SIMD: Non-SIMD load instruction	Store SIMD: Single vector contiguous store instruction	Multiple vector contiguous structure store instruction	Non-contiguous scatter store instruction	Floating-point register spill instruction	Predicate register spill instruction	Non-SIMD: Non-SIMD store instruction
0	0	3.92E+07	0.00E+00	0.00E+00	0.00E+00	1.95E+06	4.93E+05	0.00E+00	9.82E+06	1.37E+07	0.00E+00	0.00E+00	1.68E+06	2.61E+04	8.14E+05
0	1	4.14E+07	0.00E+00	0.00E+00	0.00E+00	2.00E+06	5.13E+05	0.00E+00	1.02E+07	1.43E+07	0.00E+00	0.00E+00	1.72E+06	2.60E+04	8.17E+05
0	2	4.40E+07	0.00E+00	0.00E+00	0.00E+00	2.01E+06	5.13E+05	0.00E+00	1.08E+07	1.51E+07	0.00E+00	0.00E+00	1.73E+06	2.60E+04	8.19E+05
0	3	4.27E+07	0.00E+00	0.00E+00	0.00E+00	1.95E+06	4.92E+05	0.00E+00	1.05E+07	1.47E+07	0.00E+00	0.00E+00	1.68E+06	2.60E+04	8.15E+05
0	4	4.32E+07	0.00E+00	0.00E+00	0.00E+00	2.00E+06	5.13E+05	0.00E+00	1.06E+07	1.48E+07	0.00E+00	0.00E+00	1.72E+06	2.60E+04	8.18E+05
0	5	4.40E+07	0.00E+00	0.00E+00	0.00E+00	2.01E+06	5.13E+05	0.00E+00	1.08E+07	1.51E+07	0.00E+00	0.00E+00	1.73E+06	2.60E+04	8.17E+05
0	6	4.27E+07	0.00E+00	0.00E+00	0.00E+00	1.95E+06	4.92E+05	0.00E+00	1.05E+07	1.47E+07	0.00E+00	0.00E+00	1.68E+06	2.60E+04	8.14E+05
0	7	4.32E+07	0.00E+00	0.00E+00	0.00E+00	2.00E+06	5.13E+05	0.00E+00	1.06E+07	1.48E+07	0.00E+00	0.00E+00	1.72E+06	2.60E+04	8.17E+05
0	8	4.40E+07	0.00E+00	0.00E+00	0.00E+00	2.01E+06	5.13E+05	0.00E+00	1.08E+07	1.51E+07	0.00E+00	0.00E+00	1.73E+06	2.60E+04	8.17E+05
0	9	4.27E+07	0.00E+00	0.00E+00	0.00E+00	1.95E+06	4.92E+05	0.00E+00	1.05E+07	1.47E+07	0.00E+00	0.00E+00	1.68E+06	2.60E+04	8.13E+05
0	10	4.15E+07	0.00E+00	0.00E+00	0.00E+00	2.00E+06	5.13E+05	0.00E+00	1.02E+07	1.43E+07	0.00E+00	0.00E+00	1.72E+06	2.60E+04	8.17E+05
0	11	4.03E+07	0.00E+00	0.00E+00	0.00E+00	2.01E+06	5.13E+05	0.00E+00	9.98E+06	1.41E+07	0.00E+00	0.00E+00	1.73E+06	2.60E+04	8.13E+05
CMG 0 total		5.09E+08	0.00E+00	0.00E+00	0.00E+00	2.38E+07	6.07E+06	0.00E+00	1.25E+08	1.75E+08	0.00E+00	0.00E+00	2.05E+07	3.12E+05	9.79E+06
		8.70E+08													

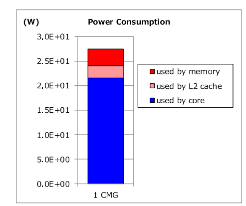
Process	Thread	Prefetch: Contiguous prefetch instruction	Gathering prefetch instruction	Scalar prefetch instruction	DCZVA instruction	Floating-point: Floating-point instruction except FMA and reciprocal	FMA instruction	Floating-point reciprocal instruction	Floating-point move and conversion: Floating-point conversion instruction	Floating-point move instruction	Integer instruction	Branch instruction	Predicate instruction	Crypto-graphic instruction	Other instruction	Total
0	0	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.13E+07	2.77E+07	0.00E+00	0.00E+00	1.16E+07	4.09E+06	5.47E+06	3.50E+06	0.00E+00	3.12E+07	1.73E+08
0	1	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.28E+07	3.02E+07	0.00E+00	0.00E+00	1.20E+07	4.47E+06	5.57E+06	3.58E+06	0.00E+00	3.27E+07	1.82E+08
0	2	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.50E+07	3.36E+07	0.00E+00	0.00E+00	1.21E+07	4.97E+06	5.57E+06	3.58E+06	0.00E+00	3.41E+07	1.94E+08
0	3	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.44E+07	3.27E+07	0.00E+00	0.00E+00	1.16E+07	4.84E+06	5.46E+06	3.50E+06	0.00E+00	3.33E+07	1.89E+08
0	4	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.44E+07	3.29E+07	0.00E+00	0.00E+00	1.20E+07	4.86E+06	5.57E+06	3.58E+06	0.00E+00	3.39E+07	1.91E+08
0	5	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.50E+07	3.36E+07	0.00E+00	0.00E+00	1.21E+07	4.97E+06	5.57E+06	3.58E+06	0.00E+00	3.42E+07	1.94E+08
0	6	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.44E+07	3.27E+07	0.00E+00	0.00E+00	1.16E+07	4.84E+06	5.46E+06	3.50E+06	0.00E+00	3.33E+07	1.89E+08
0	7	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.44E+07	3.29E+07	0.00E+00	0.00E+00	1.20E+07	4.86E+06	5.57E+06	3.58E+06	0.00E+00	3.38E+07	1.91E+08
0	8	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.50E+07	3.36E+07	0.00E+00	0.00E+00	1.21E+07	4.97E+06	5.57E+06	3.58E+06	0.00E+00	3.42E+07	1.94E+08
0	9	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.44E+07	3.27E+07	0.00E+00	0.00E+00	1.16E+07	4.84E+06	5.46E+06	3.50E+06	0.00E+00	3.34E+07	1.89E+08
0	10	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.29E+07	3.03E+07	0.00E+00	0.00E+00	1.20E+07	4.48E+06	5.57E+06	3.58E+06	0.00E+00	3.26E+07	1.83E+08
0	11	0.00E+00	0.00E+00	2.01E+03	1.60E+04	2.19E+07	2.84E+07	0.00E+00	0.00E+00	1.21E+07	4.20E+06	5.57E+06	3.58E+06	0.00E+00	3.16E+07	1.77E+08
CMG 0 total		0.00E+00	0.00E+00	2.41E+04	1.92E+05	2.86E+08	3.81E+08	0.00E+00	0.00E+00	1.43E+08	5.64E+07	6.64E+07	4.26E+07	0.00E+00	3.98E+08	2.24E+09
		2.41E+04			1.92E+05	6.67E+08			1.43E+08		5.64E+07	6.64E+07	4.26E+07	0.00E+00	3.98E+08	2.24E+09

Power Consumption (W) Process	Thread	Power consumption used by core	Power consumption used by L2 cache	Power consumption used by memory
0	0	1.74E+00	2.52E+00	3.37E+00
0	1	1.73E+00		
0	2	1.77E+00		
0	3	1.78E+00		
0	4	1.78E+00		
0	5	1.77E+00		
0	6	1.82E+00		
0	7	1.79E+00		
0	8	1.78E+00		
0	9	1.86E+00		
0	10	1.85E+00		
0	11	1.83E+00		
CMG 0 total		2.15E+01	2.52E+00	3.37E+00

Hardware Prefetch Rate (%) (/Hardware Prefetch) Process	Thread	L1: Stream mode prefetch rate	L1: Injection mode allocate prefetch rate	L1: Injection mode unallocate prefetch rate	L2: Stream mode prefetch rate	L2: Injection mode allocate prefetch rate	L2: Injection mode unallocate prefetch rate	L1/L2: Other hardware prefetch
0	0	33.22%	0.00%	0.00%	30.26%	0.00%	0.00%	36.52%
0	1	34.28%	0.00%	0.00%	30.84%	0.00%	0.00%	34.88%
0	2	33.89%	0.00%	0.00%	30.63%	0.00%	0.00%	35.48%
0	3	32.73%	0.00%	0.00%	31.16%	0.00%	0.00%	36.10%
0	4	33.45%	0.00%	0.00%	31.02%	0.00%	0.00%	35.53%
0	5	33.06%	0.00%	0.00%	30.68%	0.00%	0.00%	36.25%
0	6	32.60%	0.00%	0.00%	30.52%	0.00%	0.00%	36.88%
0	7	33.14%	0.00%	0.00%	30.15%	0.00%	0.00%	36.71%
0	8	33.53%	0.00%	0.00%	30.15%	0.00%	0.00%	36.32%
0	9	33.05%	0.00%	0.00%	31.30%	0.00%	0.00%	35.65%
0	10	35.55%	0.00%	0.00%	30.11%	0.00%	0.00%	34.33%
0	11	37.69%	0.00%	0.00%	30.07%	0.00%	0.00%	32.24%
CMG 0 total		33.85%	0.00%	0.00%	30.57%	0.00%	0.00%	35.57%

FLOPS Process	Thread	Double precision floating-point operation	Single precision floating-point operation	Half precision floating-point operation	GFLOPS by Active element rate
0	0	0.00.E+00	1.23.E+09	0.00.E+00	7.95
0	1	0.00.E+00	1.33.E+09	0.00.E+00	8.47
0	2	0.00.E+00	1.47.E+09	0.00.E+00	9.41
0	3	0.00.E+00	1.44.E+09	0.00.E+00	9.52
0	4	0.00.E+00	1.44.E+09	0.00.E+00	9.25
0	5	0.00.E+00	1.47.E+09	0.00.E+00	9.18
0	6	0.00.E+00	1.44.E+09	0.00.E+00	9.11
0	7	0.00.E+00	1.44.E+09	0.00.E+00	8.92
0	8	0.00.E+00	1.47.E+09	0.00.E+00	8.96
0	9	0.00.E+00	1.44.E+09	0.00.E+00	8.82
0	10	0.00.E+00	1.34.E+09	0.00.E+00	8.75
0	11	0.00.E+00	1.26.E+09	0.00.E+00	8.88
CMG 0 total		0.00.E+00	1.68.E+10	0.00.E+00	107.24

Extra Process	Thread	Gather instruction rate (%): 0 flow rate (%)	1 flow rate (%)	2 flows rate (%)	Instruction: Micro-operation instruction	Element manipulated instruction	Register manipulated instruction	MOVPRFX instruction	Math functional instruction	Micro decomposition instruction rate (%)	Branch prediction miss rate (%)
0	0	0.00%	0.00%	0.00%	1.80E+08	7.80E+06	3.35E+06	0.00E+00	0.00E+00	99.65%	3.54%
0	1	0.00%	0.00%	0.00%	1.90E+08	8.07E+06	3.43E+06	0.00E+00	0.00E+00	99.65%	4.37%
0	2	0.00%	0.00%	0.00%	2.02E+08	8.05E+06	3.43E+06	0.00E+00	0.00E+00	99.67%	4.15%
0	3	0.00%	0.00%	0.00%	1.97E+08	7.80E+06	3.35E+06	0.00E+00	0.00E+00	99.67%	4.26%
0	4	0.00%	0.00%	0.00%	1.99E+08	8.07E+06	3.43E+06	0.00E+00	0.00E+00	99.67%	4.11%
0	5	0.00%	0.00%	0.00%	2.02E+08	8.05E+06	3.43E+06	0.00E+00	0.00E+00	99.67%	4.25%
0	6	0.00%	0.00%	0.00%	1.97E+08	7.80E+06	3.35E+06	0.00E+00	0.00E+00	99.67%	4.16%
0	7	0.00%	0.00%	0.00%	1.99E+08	8.07E+06	3.43E+06	0.00E+00	0.00E+00	99.67%	4.26%
0	8	0.00%	0.00%	0.00%	2.02E+08	8.05E+06	3.43E+06	0.00E+00	0.00E+00	99.68%	3.86%
0	9	0.00%	0.00%	0.00%	1.97E+08	7.80E+06	3.35E+06	0.00E+00	0.00E+00	99.67%	4.13%
0	10	0.00%	0.00%	0.00%	1.90E+08	8.07E+06	3.43E+06	0.00E+00	0.00E+00	99.65%	4.60%
0	11	0.00%	0.00%	0.00%	1.84E+08	8.05E+06	3.43E+06	0.00E+00	0.00E+00	99.64%	4.21%
CMG 0 total		0.00%	0.00%	0.00%	2.34E+09	9.57E+07	4.08E+07	0.00E+00	0.00E+00	99.66%	4.16%
		0.00%			2.34E+09	1.36E+08		0.00E+00	0.00E+00	99.66%	4.16%



Data Transfer CMGs		Destination (GB/s): Own memory	Other memory	Tofu	PCI
CMG 0 total	read	2.15E+01	0.00E+00	3.21E+00	0.00E+00
	write	7.63E+00	3.58E-01	3.17E+00	0.00E+00

- 出力例：50インチぐらいの4K TVで見るとよい

Nek5000/RS axhelmカーネル

- スペクトル要素方
  - 六面体要素の中に8^3=512個の点がある
  - 各軸の8点は選点（直行多項式の根）になっている
  - 各軸に8×8行列を掛けることで勾配が求まる
    （7次の多項式になっていると思えばわかる）
- やっている計算

$$A = \begin{pmatrix} D_1 \\ D_2 \\ D_3 \end{pmatrix}^T \begin{pmatrix} G_{11} & G_{12} & G_{13} \\ G_{12} & G_{22} & G_{23} \\ G_{13} & G_{23} & G_{33} \end{pmatrix} \begin{pmatrix} D_1 \\ D_2 \\ D_3 \end{pmatrix} \qquad (\nabla\vec{v}, \nabla\vec{u}) = \vec{v}^T A \vec{u}$$

  - 勾配、幾何係数、勾配の転地の順に掛ける
  - 幾何係数：六面体の中の座標での微分を外の座標での微分に直す（＋ヤコビアンと密度で重み付け）
  - 勾配は演算ネック、幾何係数はメモリネック（6語読んで9乗算）

As-isとtune-1

```
79        for(int k = 0; k < p_Nq; ++k) for(int j = 0; j < p_Nq; ++j) // loop-2
80        {
81            for(int i = 0; i < p_Nq; ++i) {
82                // const dlong id = i + j * p_Nq + k * p_Nq * p_Nq + element * p_Np;
83                const dlong gbase = element * p_Nggeo * p_Np + k * p_Nq * p_Nq + j * p_Nq + i;
84                const dfloat r_G00 = ggeo[gbase + p_G00ID * p_Np]; // #define p_G00ID 1
85                const dfloat r_G01 = ggeo[gbase + p_G01ID * p_Np]; // #define p_G01ID 2
86                const dfloat r_G11 = ggeo[gbase + p_G11ID * p_Np]; // #define p_G11ID 3
87                const dfloat r_G12 = ggeo[gbase + p_G12ID * p_Np]; // #define p_G12ID 4
88                const dfloat r_G02 = ggeo[gbase + p_G02ID * p_Np]; // #define p_G02ID 5
89                const dfloat r_G22 = ggeo[gbase + p_G22ID * p_Np]; // #define p_G22ID 6
90
91                dfloat qr = 0.f;
92                dfloat qs = 0.f;
93                dfloat qt = 0.f;
94                // loop-2a (gradient)
95                for(int m = 0; m < p_Nq; ++m) {
96                    qr += s_D[i][m] * s_q[k][j][m];
97                    qs += s_D[j][m] * s_q[k][m][i];
98                    qt += s_D[k][m] * s_q[m][j][i];
99                }
00                // loop-2b (geometry factor)
01                s_Gqr[k][j][i] = r_G00 * qr + r_G01 * qs + r_G02 * qt;
02                s_Gqs[k][j][i] = r_G01 * qr + r_G11 * qs + r_G12 * qt;
03                s_Gqt[k][j][i] = r_G02 * qr + r_G12 * qs + r_G22 * qt;
04            }
```

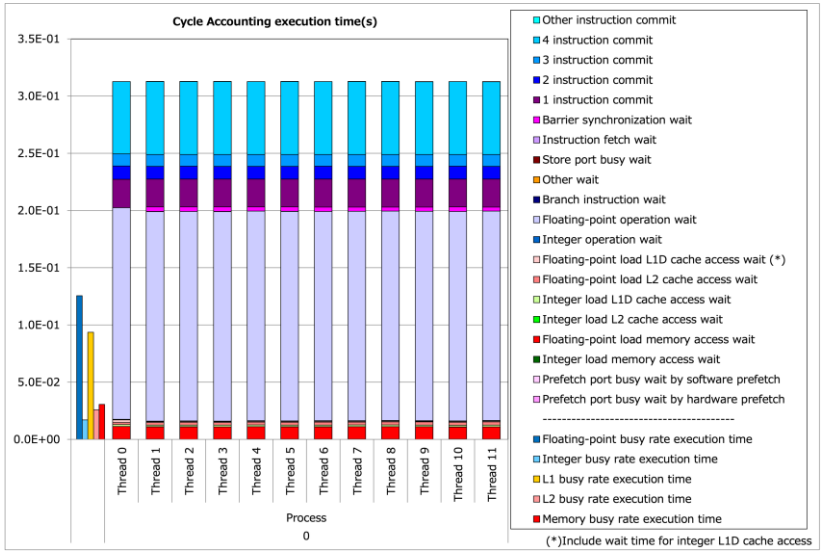
- 最内ループ変数mに対するSIMD化では不適切である
  - Gatherと水平加算の発生
  - 幾何係数の部分が一切SIMD化されない

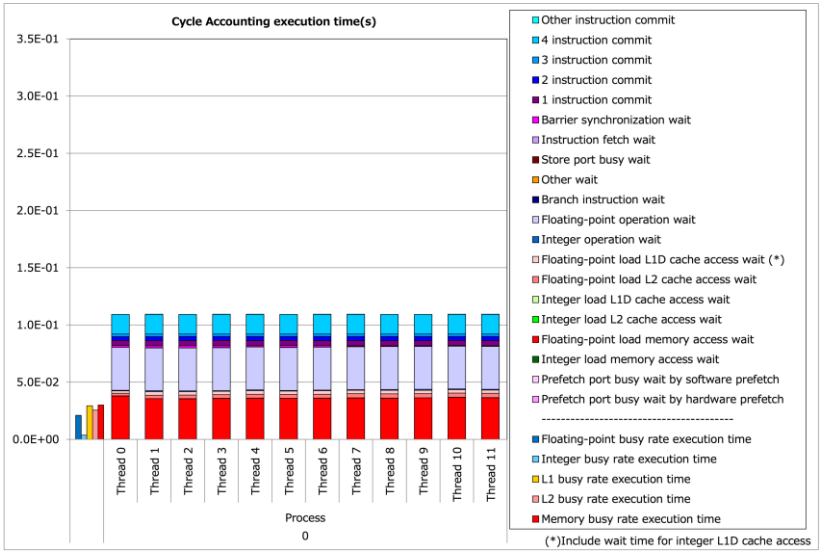
```
jwd8222o-i  "axhelm-0.cpp", line 78: このループで必要なプリフェッチの数が、ハードウェアプリフェッチの許容数
jwd8662o-i  "axhelm-0.cpp", line 78: スケジューリング結果を得られなかったため、ソフトウェアパイプライニング
jwd8203o-i  "axhelm-0.cpp", line 80: このループをフルアンローリングしました。
jwd6004s-i  "axhelm-0.cpp", line 94: リダクション演算を含むループ制御変数'm'のループをSIMD化しました。
jwd8209o-i  "axhelm-0.cpp", line 100: 多項式の演算順序を変更しました。
jwd8668o-i  "axhelm-0.cpp", line 106: 命令数が多いため、ソフトウェアパイプライニングを抑止しました。
jwd8203o-i  "axhelm-0.cpp", line 106: このループをフルアンローリングしました。
jwd8203o-i  "axhelm-0.cpp", line 108: このループをフルアンローリングしました。
jwd6004s-i  "axhelm-0.cpp", line 112: リダクション演算を含むループ制御変数'm'のループをSIMD化しました。
```

```
94 #pragma loop fullunroll_pre_simd
95                 for(int m = 0; m < p_Nq; ++m) {
96                     qr += s_D[i][m] * s_q[k][j][m];
97                     qs += s_D[j][m] * s_q[k][m][i];
98                     qt += s_D[k][m] * s_q[m][j][i];
99                 }
```

- ディレクティブでm-loopをアンロールするとひとつ外側のループ変数iに対してSIMD化される

Before/after

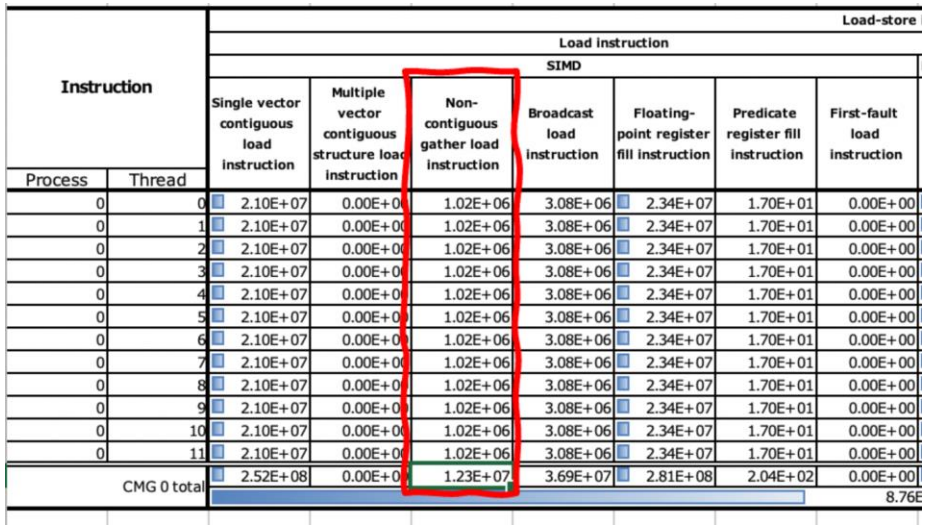




- 効果は抜群だが、、、
- 左側のPA情報からこのチューニングに到れるかどうかが問題
- どのループ変数でSIMD化したかはコンパイラのメッセージからわかる
  - mでは不適切でiならよいと気付くことができるか？
- ディレクティブは知らないかもしれない
  - mとiのループ順序を入れ替えることでも対処は可能

Tune-2 (ストライドアクセスの回避)

```
95                  for(int m = 0; m < p_Nq; ++m) {
96                      qr += s_D[i][m] * s_q[k][j][m];
97                      qs += s_D[j][m] * s_q[k][m][i];
98                      qt += s_D[k][m] * s_q[m][j][i];
99                  }
```



Instruction		Load-store						
		Load instruction						
		SIMD						
Process	Thread	Single vector contiguous load instruction	Multiple vector contiguous structure load instruction	Non-contiguous gather load instruction	Broadcast load instruction	Floating-point register fill instruction	Predicate register fill instruction	First-fault load instruction
0	0	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	1	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	2	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	3	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	4	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	5	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	6	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	7	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	8	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	9	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	10	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
0	11	2.10E+07	0.00E+00	1.02E+06	3.08E+06	2.34E+07	1.70E+01	0.00E+00
CMG 0 total		2.52E+08	0.00E+00	1.23E+07	3.69E+07	2.81E+08	2.04E+02	0.00E+00
								8.76E

```
97                  for(int m = 0; m < p_Nq; ++m) {
98                      qr += s_T[m][i] * s_q[k][j][m];
99                      qs += s_D[j][m] * s_q[k][m][i];
100                     qt += s_D[k][m] * s_q[m][j][i];
101                 }
```

- 8×8の行列s_Dは要素に非依存
- Gather/scatterの発生はプロファイラで検出可能
- 関数の入り口で転置行列s_Tを作っておく

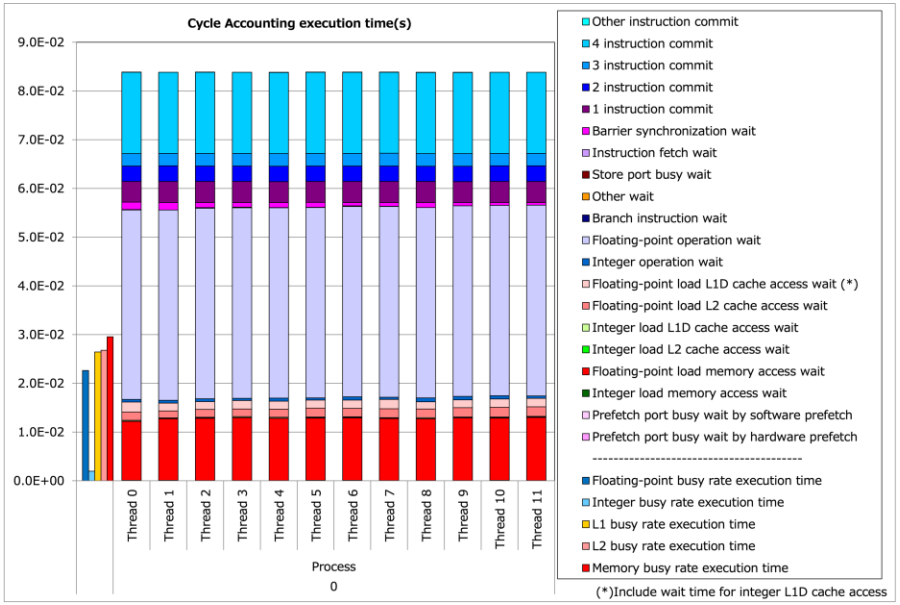
Tune-3 （ソフトウェアパイプラン）

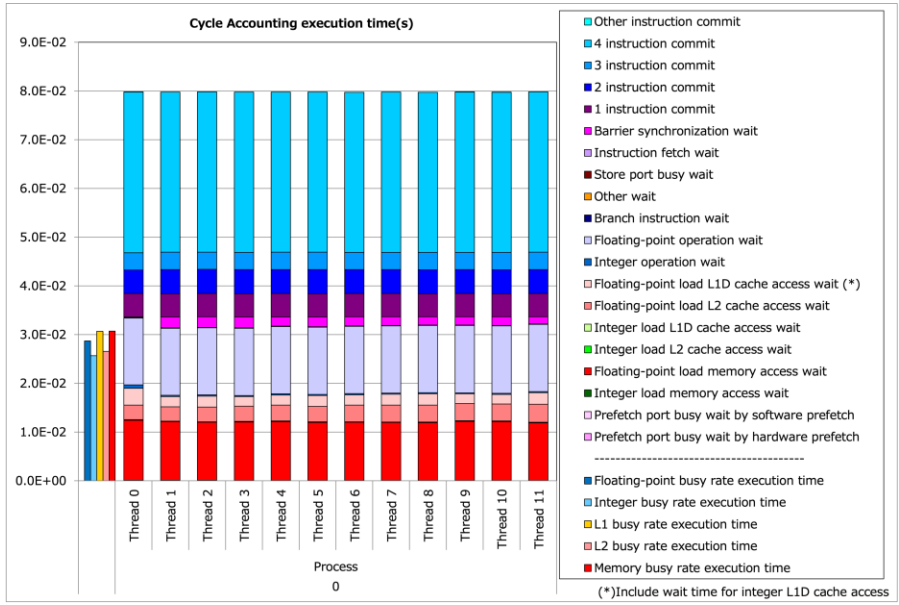
- mループではフルアンロール、iループはSIMD化
- その外側にj、kとループがあるが長さが8でソフトウェアパイプライニングを適用するには短い
  - 現状ではjについてフルアンロールされている
- k-loopとj-loopを手動でcollapseさせると長さが64になりswpが有効になる
  - OpenMPのcollapseは同時にスレッド並列化もするので使えない

```
84          for(int kj = 0; kj < p_Nq*p_Nq; ++kj) {
85              const int j = kj % p_Nq;
86              const int k = kj / p_Nq;
```

```
jwd8204o-i  "axhelm-3.cpp", line 73: ループにソフトウェアパイプライニングを適用しました。
jwd8205o-i  "axhelm-3.cpp", line 73: ループの繰返し数が64回以上の時、ソフトウェアパイプライニングを適用した
jwd6001s-i  "axhelm-3.cpp", line 77: ループ制御変数'i'のループをSIMD化しました。
jwd8204o-i  "axhelm-3.cpp", line 84: ループにソフトウェアパイプライニングを適用しました。
jwd8205o-i  "axhelm-3.cpp", line 84: ループの繰返し数が48回以上の時、ソフトウェアパイプライニングを適用した
jwd6001s-i  "axhelm-3.cpp", line 88: ループ制御変数'i'のループをSIMD化しました。
jwd8203o-i  "axhelm-3.cpp", line 103: このループをフルアンローリングしました。
jwd8204o-i  "axhelm-3.cpp", line 115: ループにソフトウェアパイプライニングを適用しました。
jwd8205o-i  "axhelm-3.cpp", line 115: ループの繰返し数が48回以上の時、ソフトウェアパイプライニングを適用し
jwd6001s-i  "axhelm-3.cpp", line 119: ループ制御変数'i'のループをSIMD化しました。
jwd8203o-i  "axhelm-3.cpp", line 124: このループをフルアンローリングしました。
```

Before/after





- 待ち時間は減った？
- IPCが1.01→2.01、コアあたりGIPSが2.03→4.02
- 答え：整数命令が増えた、実行時間の減少は少しだけ

プリフェッチ

幾何係数の部分のデータ構造
ggeo[Nelement][7][512]

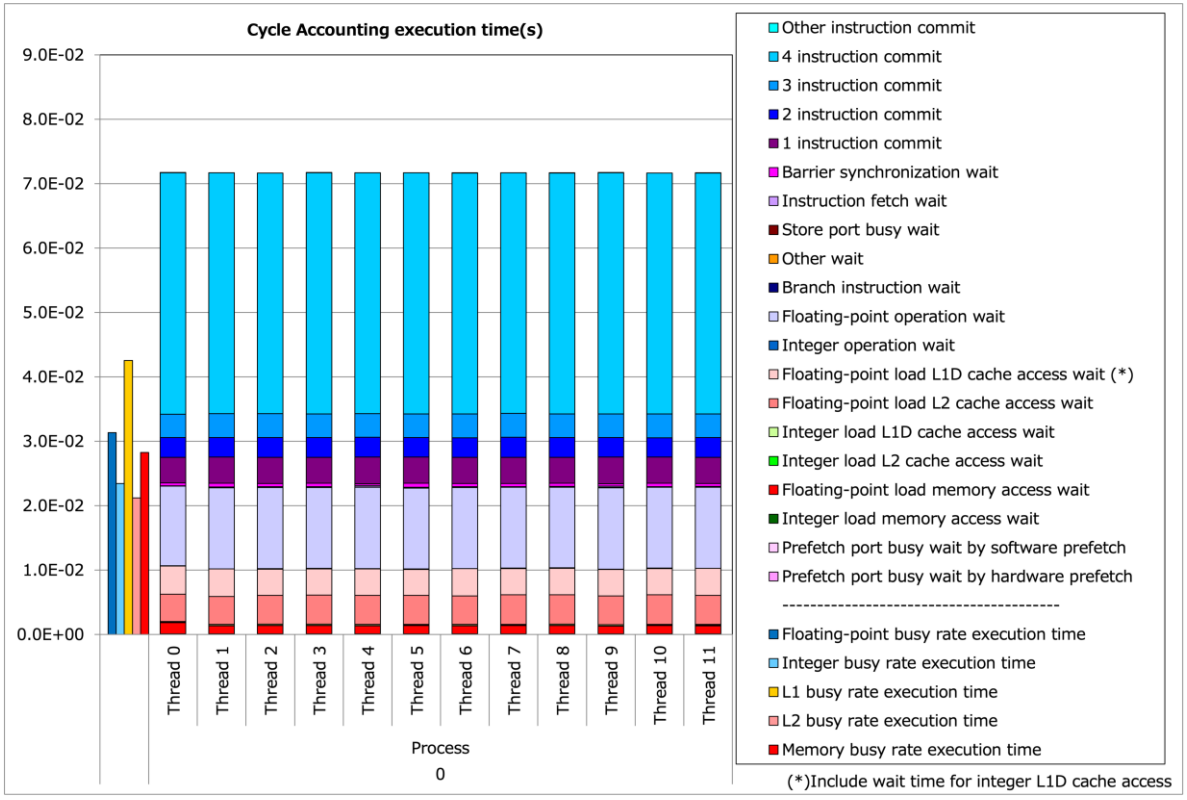
- 冗長なアドレス計算を手で取り除く
  - kとjを求めてからアドレス計算→kjから直接アドレス計算

```
84        for(int kj = 0; kj < p_Nq*p_Nq; ++kj) {
85            const int j = kj % p_Nq;
86            const int k = kj / p_Nq;
87
88            for(int i = 0; i < p_Nq; ++i) {
89                // const dlong id = i + j * p_Nq + k * p_Nq * p_Nq + element * p_Np;
90                const dlong gbase = element * p_Nggeo * p_Np + k * p_Nq * p_Nq + j * p_Nq + i;
```

```
84        for(int kj = 0; kj < p_Nq*p_Nq; ++kj) {
85                    const int j = kj % p_Nq;
86                    const int k = kj / p_Nq;
87
88            for(int i = 0; i < p_Nq; ++i) {
89                // const dlong id = i + j * p_Nq + k * p_Nq * p_Nq + element * p_Np;
90                const dlong gbase = element * p_Nggeo * p_Np + kj * p_Nq + i;
```

- 急に速くなった
- アセンブリを見るとプリフェッチ命令が出るようになっていた
  - コンパイラからのメッセージはなし
  - PA情報からキャッシュミス情報を見ることでもわかったかも

最終版 (swp+prefetch)



- L1ビジーが大きい（59.33%）
- 演算：ピークの19.1%
- メモリスループット：ピークの39.4%

axhelmまとめ

- プロファイラでわかること
  - メモリ待ち時間
  - 演算待ち時間
  - Gather/Scatter命令
    - ソースコード上の場所までは示してくれない
  - （スレッド間インバランス）
- チューニングの指針となったか？
  - 消したい待ち時間はある
  - 「このスイッチをオンにしたら待ち時間が消えます」という便利なものはない（-Kfastオプションから始めるなら）
  - 結局は試行錯誤による発見的手法になる
  - Before/afterの比較が情報量に富む

Bridge++ QXS (格子QCD)

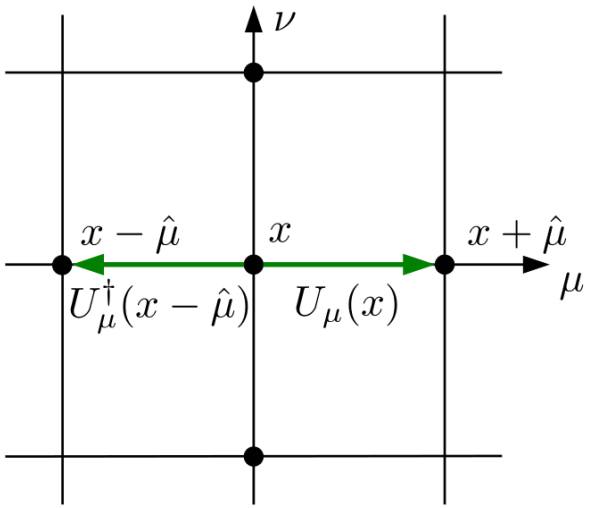
- SIMDを2次元方向に使おうという実装
  - float16のSIMDなら例えばxy方向に4×4で
- Even-odd法との組み合わせ
- C++でtemplatを多用しておりコンパイルはClangモード
  - ACLE for SVEも多用

- 「正しく計算できるけれどflops値がいまいちなので」と金森さんから調査と改善の依頼

格子QCDの計算

$$D_{x,y} = [1+F(x)]\delta_{x,y} - \kappa \sum_{\mu=1}^{4} \left[(1-\gamma_\mu) U_\mu(x) \delta_{x+\hat{\mu},y} + (1+\gamma_\mu) U_\mu^\dagger(x-\hat{\mu}) \delta_{x-\hat{\mu},y} \right],$$

- 格子点上にクォークのスピン
  （Dirac spinが3色で複素12語）
- リンク変数にゲージ（グルーオン）
  4方向に3×3ユニタリ行列
- 9点（対角+8点）のステンシル計算



Naïveなデータ構造

```
real spin [NT][NZ][NY][NX][3][4][2];
real gauge[NT][NZ][NY][NX][4][3][3][2];
```

格子点あたり24語と72語、floatなら96 byteと144 byte

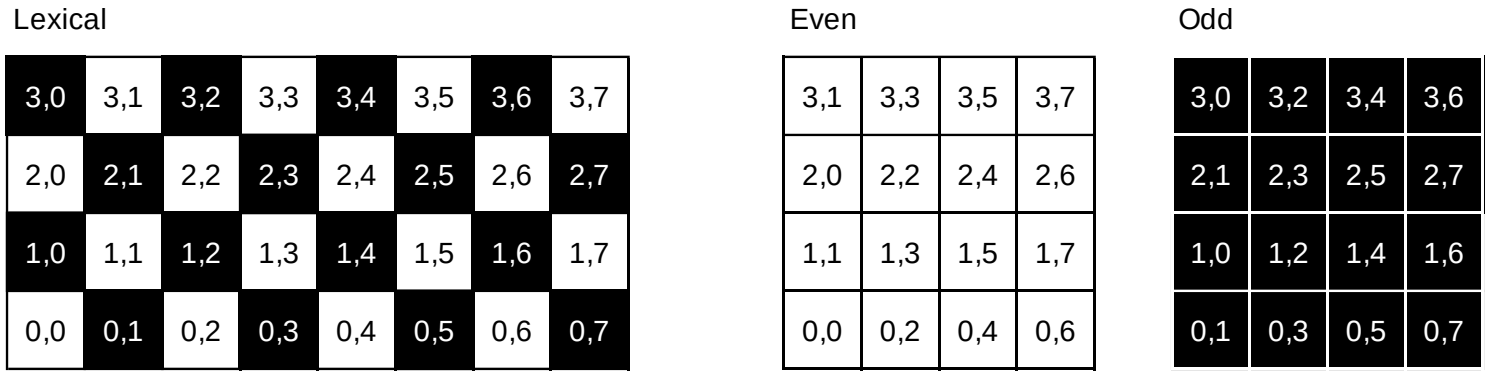
γ 行列

- $(1 \pm \gamma_\mu)$を掛けるとスピンの自由度が複素4語から複素2語に減るのに注意

$$\gamma_1 = \begin{pmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{pmatrix} \quad \gamma_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad \gamma_3 = \begin{pmatrix} 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \\ -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \quad \gamma_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

(42)

Even-odd法



$$\begin{pmatrix} D_{ee} & D_{eo} \\ D_{oe} & D_{oo} \end{pmatrix} \begin{pmatrix} x_e \\ x_o \end{pmatrix} = \begin{pmatrix} b_e \\ b_o \end{pmatrix}$$

$$\underbrace{\left(1 - D_{ee}^{-1} D_{eo} D_{oo}^{-1} D_{oe}\right)}_{\equiv D} x_e = D_{ee}^{-1}\left(b_e - D_{eo} D_{oo}^{-1} b_o\right)$$

$$x_o = D_{oo}^{-1}\left(b_o - D_{oe} x_e\right)$$

- D_{oe}: Even成分を読んでOdd成分に書き込み⇔ D_{eo}
- D_{ee},D_{oo}：Wilsonカーネルでは単なる対角行列
- 1回のmultの演算数は減らないが2 hopするので収束が早い
- CG法等で複数のベクトルを保持するメモリフットプリントが半分

QWSとQXS

- QWS: QCD Wide SIMD
  - 重点課題9のターゲットアプリ
  - 単精度16語のSIMDとeven-odd法の組み合わせにより、MPIプロセス当たりのx軸の長さが**32の倍数**である必要
  - 目標の192^4格子には使えるが、、、
- Bridge++ QXS
  - SIMDの軸を2次元化
  - x軸とy軸の4×4タイルをSIMDのレジスタに載せる
  - その分x方向とy方向のステンシルアクセスが非自明に

例えばこんなレイアウトに

```
float spin_eo [NT][NZ][NY/4][NX/8] [3][4]   [2] [16];
float gauge_eo[NT][NZ][NY/4][NX/8] [4][3][3][2] [16];
```

SIMD命令の複素数サポートについて

- 一部のSIMD命令では偶奇のレーンを束ねての複素数演算をサポートしている
  - BlueGeneや京、FX100（倍精度限定）
  - x86はshuffl系の命令を多用する
  - 第1オペランドの偶数レーン複製
  - 第1オペランドの奇数レーン複製
  - 第2オペランドの偶奇スワップと符号変更
- これがあると格子QCDの計算で嬉しい
  - 実効的にベクトル長が半分になる
  - 実効的にレジスタの本数が倍になる
    - スピンの12本をレジスタに置きっぱなしにできる
    - ゲージの9本をレジスタに置きっぱなしにできる (DWF)
- SVEの命令セットには入ったがA64FXではマイクロコード実装

ステンシルアクセス: X-

			3,-1	3,1	3,3	3,5	3,7
				2,0	2,2	2,4	2,6
			1,-1	1,1	1,3	1,5	1,7
				0,0	0,2	0,4	0,6

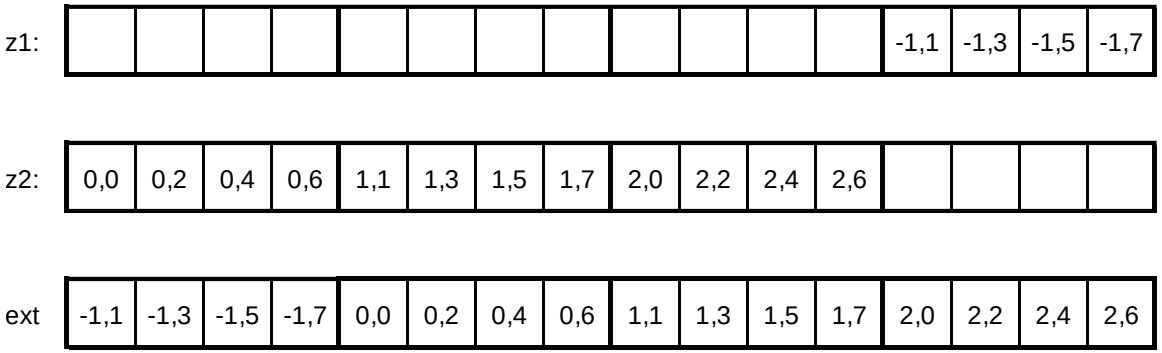
3,0	3,2	3,4	3,6
2,1	2,3	2,5	2,7
1,0	1,2	1,4	1,6
0,1	0,3	0,5	0,7

z1:	0,0	0,2	0,4	0,6	1,1	1,3	1,5		2,0	2,2	2,4	2,6	3,1	3,3	3,5	
z2:								1,-1								3,-1
pg:	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
sel:	0,0	0,2	0,4	0,6	1,1	1,3	1,5	1,-1	2,0	2,2	2,4	2,6	3,1	3,3	3,5	3,-1
idx:	0	1	2	3	7	4	5	6	8	9	10	11	15	12	13	14
tbl:	0,0	0,2	0,4	0,6	1,-1	1,1	1,3	1,5	2,0	2,2	2,4	2,6	3,-1	3,1	3,3	3,5

- Oddのひとつ左（X-）のevenからの読み込み（右にシフトして足す）
- yが奇数の行だけ左にずれる
- load, load, sel, tblで実現可能

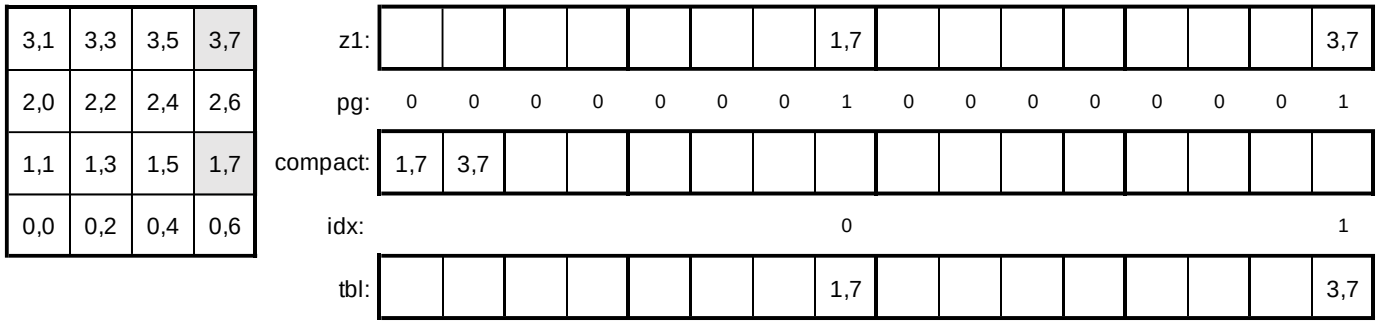
ステンシルアクセス: Y-

3,1	3,3	3,5	3,7
2,0	2,2	2,4	2,6
1,1	1,3	1,5	1,7
0,0	0,2	0,4	0,6
-1,1	-1,3	-1,5	-1,7



- load, load, extでできる
- ext: 2つのベクトルを連結して即値でシフト

通信バッファのgather/scatte回避

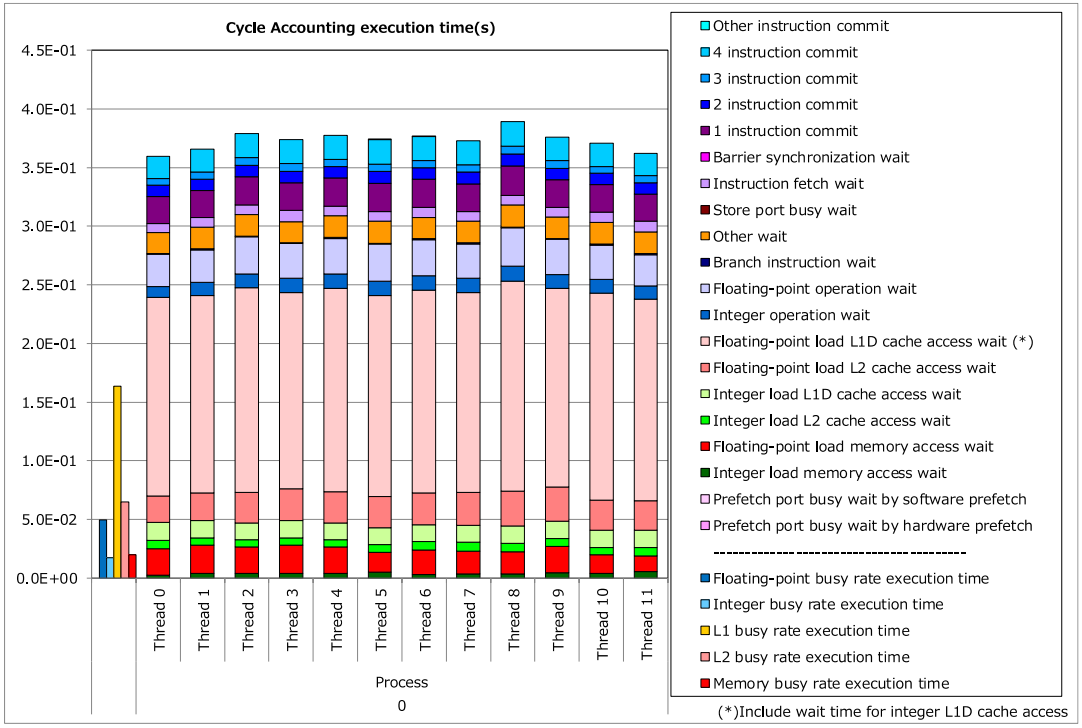


- 16語中2語だけ送信バッファに詰める (compact)
  - レジスタ内で連続にしscatterを回避
- 受信バッファから元の場所に戻す (tbl)
  - Gatherを回避

計測条件、チューニング

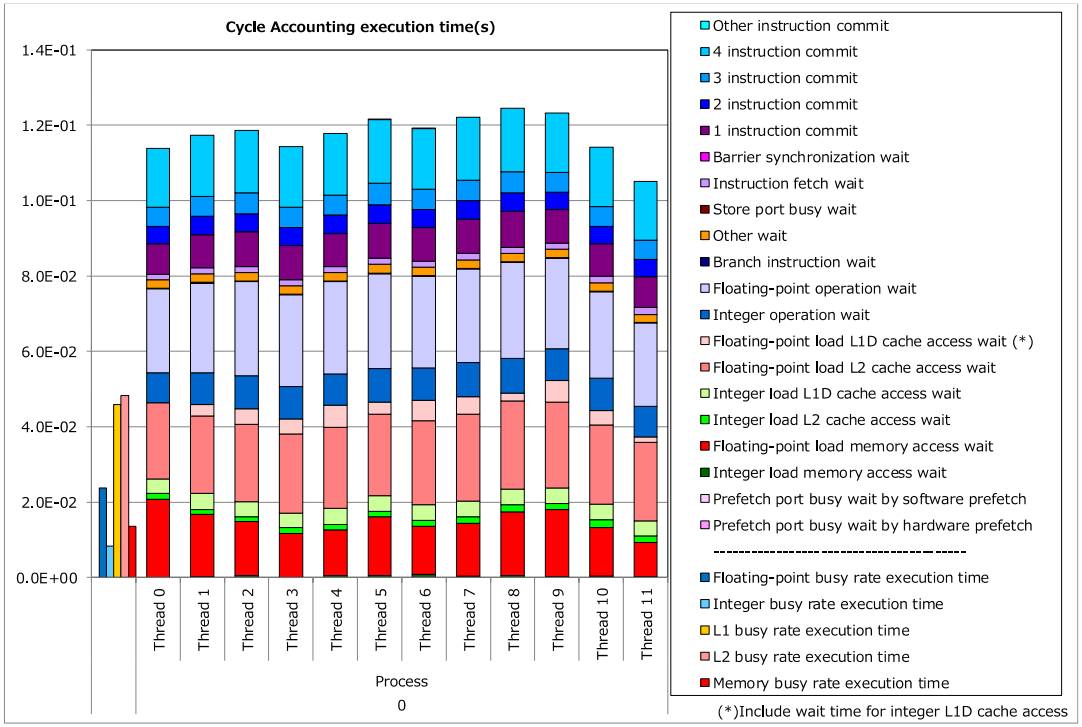
- A64FX 1ノード4プロセス、16^4=64K格子
  - CMGあたり8×8×16×16
  - SIMDとEOも含めると 8×8×4×2
- カーネルとなる関数
  - eo1: 送信バッファの準備
    - 途中でスレッドの同期とMPI送信のキック
  - bulk: ローカルでできる計算
    - スレッドを同期してMPIの受信待ち
  - eo2: 受信バッファからの計算
- コンパイラオプション
  - -Rpass-missed=inline -mllvm -inline-threshold=1000 を追加
  - 8方向からのhopのカーネルで関数callがなくなるように

As-is状態



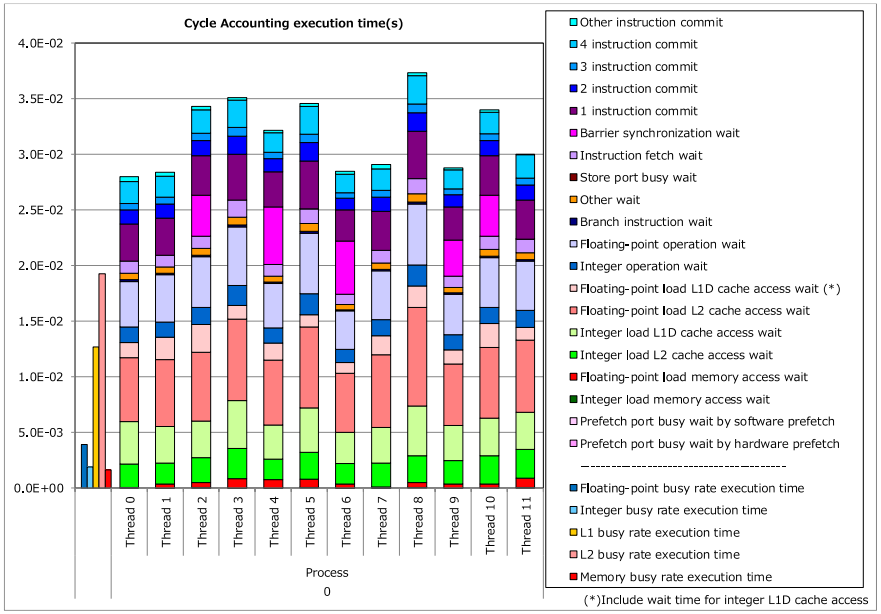
- Bulk部分
- L1アクセス待ち、L1ビジー率が高い
- Gather/scatterが沢山出ていた
  - 実は対角の部分、最後にκを掛けてストアするだけの部分
  - Vsimd_tというSIMDエミュレーション型
  - といっても単なる外側24、内側16の二重ループなのだが

チューン（性能デバッグ）後

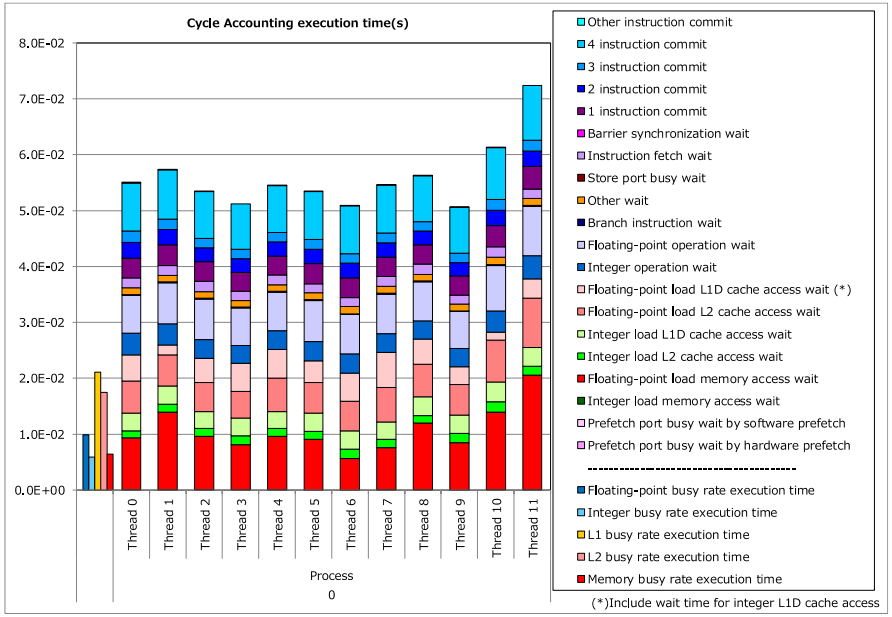


- Bulk
- 普通の見た目になっている
- 180 Gflops→360 Gflops
- 若干主記憶アクセスが発生しているのが気になる
  - 6MiB/CMGぐらいのはず
- 若干のロードインバランス

ロードバランス問題



eo1 (export)



eo2 (import)

- 実行時間／演算数含め可視化してくれる
- 格子点数均等割にしているので、袖に触る／触らないで負荷バランスが崩れる

まとめ（プロファイラで何が見えたか）

- 手を動かせば必ず絵が出てくるのが有難い
- （意図せぬ）Gather/Scatter
- ロードインバランス

- 見ておくべきもの
  - ビジー率
  - ある程度のチューンができたコードは大抵複数のリソースが忙しい
  - 「実アプリだから主記憶帯域ネック」は本当？

- 見えにくいもの
  - ソースコードの何処でそれが起こっているか？
    （区間の細分化には限度がある）
  - アセンブリ出力を確認することの代替にはならない