Optimization of stencil-based fusion kernels on Tera-flops many-core architectures

Y. ASAHI[1], G. Latu[2], T. Ina[1], Y. Idomura[1], V. Grandgirard[2], X. Garbet[2]
Japan Atomic Energy Agency (JAEA)[1], Commissariat à l'énergie atomique et aux énergies alternatives (CEA), Cadarache[2]

Abstract

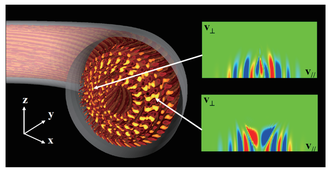
Plasma turbulence is of great importance in fusion science. Since plasma turbulence are described by 5D gyrokinetic model, Exa-scale computing is needed for simulating the next generation fusion device. Thanks to communication overlap techniques, strong scaling of stencil-based gyrokinetic codes was extended up to 100k nodes [8], and the remaining critical issue towards Exa-scale computing is to develop optimization techniques for many core architectures. This motivates us to develop advanced algorithms and optimizations of stencil-based fusion codes on Tera-flops many-core architectures like Xeon Phi, Nvidia Tesla and Fujitsu FX100.

The objective of this study is:

- to figure out the bottlenecks of a stencil-based fusion code on many-core architectures
- to establish the optimization strategy for that

GYSELA and GT5D kernels

- Both GYSELA [1] and GT5D [2] solve **plasma turbulence** in 5D phase space (x, y, z, $v_{\parallel}$, $v_{\perp}$) based on gyrokinetic model [3]



Each grid point has structure in real space (x, y, z) and velocity space ($v_{\parallel}$, $v_{\perp}$)

- **4D convection operator** in the gyrokinteic equation

$$\frac{\partial f}{\partial t} = \mathcal{G}(f) + \mathcal{C}(f) \qquad \mathcal{G}(f) = -\mathbf{U}_1 \cdot \frac{\partial f}{\partial \mathbf{R}} - U_2 \frac{\partial f}{\partial v_{\parallel}}$$ **4D Op.**

GYSELA kernel computes 4D convection operator (split into 1D+1D +2D parts) with **Semi-Lagrangian** scheme (1D or 2D cubic spline).

GT5D kernel computes the 4D convection operator with a **finite difference** (4th order Morinishi scheme, 17-stencils).

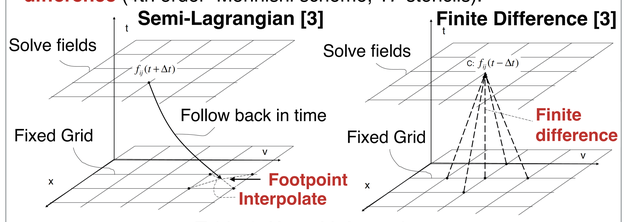


Table 1. Kernel Information

	GYSELA [1]	GT5D [2]
Numerical Scheme	Semi-Lagrangian	Finite-difference
Kernel	2D Cubic Spline	4D FD (17-stencil)
Problem size	(128, 72, 52, 201)	(128, 16, 32, 32)
Memory access	1 load/store + 2 load	1 load/store + 6 load
Flop	87	63
Byte / Flops (b/f)	0.37	1.01

Architectures

Note: Memory access and Flop in single iteration are shown

Table 2. Many-core architectures

	SB	FX100 [4]	Phi [5]	K20X [6]
CPU / GPU	SandyBridge-EP	SPARC64XIfx	Xeon Phi 5110P	Tesla K20X
Compiler	intel compiler 13.1.3	Fujitsu compiler 2.0.0	intel compiler 13.1.3	pgfortran 15.1 nvcc 6.5
Cores	8	32 + 2	60	896
Cache [MB]	20	24	0.5 x 60	1.5
Memory [GB]	64	32	8	6
Peak GFlops	172.8	1000	1010	1310
Peak B/W [GB/s]	51.2	480	320	250
SIMD width	4 (AVX)	4	8	N/A
B/F ratio	0.3	0.5	0.3	0.19

Optimization of GYSELA kernel

Algorithm 1 2D advection with Semi-Lagrangian scheme

for *All grid points* $(x_i, v_{\parallel})$ **do**
  $\eta(y=*, z=*) \leftarrow$ spline coeff. from the 2D function $f^n(x_i, y=*, z=*, v_{\parallel})$;
  for *All grid points* (y_j, z_k) **do**
    $(y_j, z_k)^* \leftarrow$ foot of characteristic for one time step Δt that ends at $(x_i, y_j, z_k, v_{\parallel})$;
    Interpolate f^n at location $(z_k)^*$ using η coeff.;
    $f^{n+1}(x_i, y_j, z_k, v_{\parallel}) \leftarrow$ the interpolated value;
  end for
end for

Optimization on SB [7] and FX100

- Optimization with data alignment, cache blocking, vectorization
- Optimization for 4 (8) SIMD width on SB, FX100 (Phi)

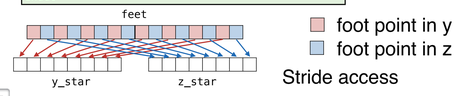
Optimization on Xeon Phi

- Realign the feet array from array of structure style (AoS) to structure of array (SoA) style

Original — Feet array stores foot point information in y direction and z direction one after the other (AoS style)

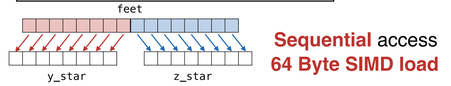
VSIZE = 8 (SIMD width)

```
do local_i = 0, VSIZE-1
  y_star(local_i) = feet(2*global_i), j, k)
  z_star(local_i) = feet(2*global_i)+1, j, k)
enddo
```



Optimized — Feet array stores foot point information in y direction sequentially followed by z direction array (**SoA** style)

```
do local_i = 0, VSIZE-1
  y_star(local_i)  = feet(2*global_i, j,k)
  z_star(local_i)  = feet(2*global_i+VSIZE,j,k)
enddo
```



- OpenMP Dynamic scheduling for efficient pipeline usage
  Even if a thread gets stuck with one type of instruction, another thread can execute other types of instructions

Note: Xeon Phi is in-order instruction processor

- Effects of Optimizations

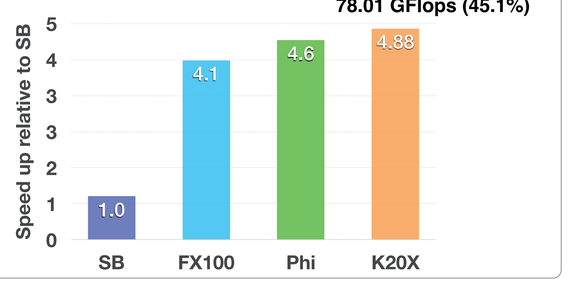
	Original	Realign	Dynamic
GFlops	250	310	**344**
Relative to peak (%)	25	31	34
Speed up	-	1.24	**1.38**

Optimization on TeslaK20X

- Apply SoA style to feet array for coalesced load
- L1 cache is used to hold 2D spline coefficients

Speed up relative to Sandy Bridge

- Applying **SoA** style for SIMD load is essential for good performance on many-core architectures.
- **Highly-localized cache usage (spline coefficients)** is a key ingredient for good performance.



Optimization of GT5D kernel

Algorithm 1 4 dimensional 4th order finite difference

for *All grid points* $(x_i, y_j, z_k, v_{\parallel})$ **do**
  $C_0(l,j,i)$, $C_1(l,j,i)$, $C_2(l,j,i) \leftarrow$ coeff. used for Morinishi scheme
  $f^{n+1}_{l,k,j,i} = C_0 f^n_{l,k,j,i} + C_1\left(f^n_{l-1,k,j,i} + f^n_{l+1,k,j,i} + f^n_{l,k-1,j,i} + f^n_{l,k+1,j,i} + f^n_{l,k,j-1,i} + f^n_{l,k,j+1,i} + f^n_{l,k,j,i-1} + f^n_{l,k,j,i+1}\right)$
    $+ C_2\left(f^n_{l-2,k,j,i} + f^n_{l+2,k,j,i} + f^n_{l,k-2,j,i} + f^n_{l,k+2,j,i} + f^n_{l,k,j-2,i} + f^n_{l,k,j+2,i} + f^n_{l,k,j,i-2} + f^n_{l,k,j,i+2}\right)$
end for

Optimization on SB [8] and FX100

- Replacing block loop decomposition to cyclic loop decomposition to reuse shared cache (SB: 2.5 MB/core, FX100: 0.75 MB/core)

```
f(-1:nv+2, -1:nz+2, -1:ny+2, -1:nx+2)
l ──────────→ 1 kB
k ──────────────────→ 21 kB
j ──────────────────────────→ 760 kB
i ──────────────────────────────────→ 27.4 MB
```

5-stencil cache usage	(block) b/f = 1.0	(cyclic) b/f = **0.5**
l 8B x 5 = 40B	→ on-cache	on-cache
k 1kB x 5 = 5kB	→ on-cache	on-cache
j 21kB x 5 = 105kB	→ on-cache	on-cache
i 760kB x 5 = 3.8MB	→ memory access	**on-cache (shared)**

Reuse the data loaded by adjacent threads

Optimization on Xeon Phi

- OpenMP loop collapse for enlarging the loop size
- Dynamic scheduling for effective pipeline usage

Optimization on TeslaK20X

- Change the dimension to allocate the block from (l, **k**) [9] to (l, **j**)

```
l = threadidx%x
  +(blockidx%x-1) * blockDim%x
k = threadidx%y
  +(blockidx%y-1) * blockDim%y
do i = 1, nx
do j = 1, ny
 if( threadidx%y == 1) then
   Ci1(threadidx%x) = A(l,j,i)
                    + A(l,j,i+1)
   Ci2(threadidx%x) = A(l,j,i)
                    + A(l,j,i-1)
 end if
 call syncthreads()
! Finite difference computation
end do
end do
```

```
l = threadidx%x
  +(blockidx%x-1) * blockDim%x
j = threadidx%y
  +(blockidx%y-1) * blockDim%y
i = blockidx%z
Ci1 = A(l,j,i)
    + A(l,j,i+1)
Ci2 = A(l,j,i)
    + A(l,j,i-1)
do k = 1, nz
! Finite difference computation
end do
```

Ci1, Ci2 : coefficients for Morinishi scheme
A : 3D const. array

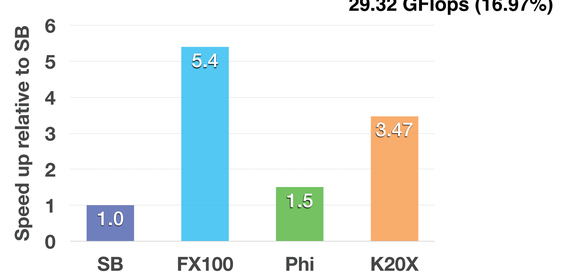
Avoid a divergence branch for coef. computation in the inner most loop

- Remapping thread block to load data and halo region with coalesced load as described in Ref. [10]
- Reuse (l, j) plane in shared memory and registers cyclically in k direction (the values with k-index from k-2 to k+2 are kept)
- Effects of Optimizations

	Original [9]	No divergence	Register reuse
GFlops	15.29	57.64	**95.58**
Relative to peak (%)	1.2	4.4	7.3
Speed up	-	3.77	**6.25**

Speed up relative to Sandy Bridge

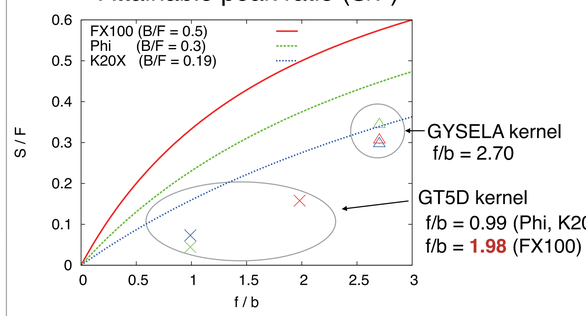
- Large shared cache plays important role on Sandy and FX100.
- In Xeon Phi, the reuse of shared cache is not effective due to large remote cache latency [11]



Discussion on the architectures

Roofline model

The obtained performance is compared with the roofline model [12] (f/b and B/F are given in Tables 1 and 2)



Note: GT5D on FX100 has larger f/b ratio

FX100

- Since FX100 has common features as Sandy Bridge (large shared cache and 4 SIMD), the **same optimization strategy** works well, and both kernels attain reasonably good performance with minimum optimization.

Xeon Phi

- Since Phi is in-order processor, the **OpenMP dynamic** scheduling improves an efficiency of pipeline usage (effective in both kernels).
- A good performance of GYSELA shows that a key ingredient for the good performance on Phi is **cache locality**. Remote cache access used in GT5D leads to performance degradation.

TeslaK20X

- In order to keep coalesced load, the **SoA style** is preferable which is also the case for Phi.
- For the 4D complex data structure, **appropriate assignment of the thread blocks** is important (for avoiding a warp divergence).

Future issues on real applications

For accelerators, remaining parts of the codes (except the studied kernels) have to be ported also. One should pay attention to:

- the limited available amount of memory
- the direct communications between accelerators

Conclusions

We optimized the kernels from fusion plasma codes, GYSELA and GT5D, on Tera-flops many-core architectures including accelerators (Phi, K20X), and multi-core CPUs (FX100). Though the comparison of optimized performance, we found optimization strategies important on accelerators which are not clear on multi-core CPUs.

- The drawbacks of in-order instruction processor can be overcome with the use of **OpenMP dynamic scheduling**.
- For effective memory access, the **structure of array (SoA)** style is preferable.
- The **high cache locality** is critical for performance, which can only be achieved with algorithm level optimization.

Reference

[1] V. Grandgirard, et al., Commun. Nonlinear Sci. Num. Simul., 13 81, 2008.
[2] Y. Idomura, et al., Comput. Phys. Commun. 179 391, 2008.
[3] X. Garbet, Y. Idomura, L. Villard and T.-H. Watanabe, Nucl. Fusion 50, 043002 (2010).
[4] INTEL, Intel xeon phi coprocessor inst. set archi. ref. manual, https://software.intel.com/en-us/mic-developer.
[5] Nvidia. TESLA K20X GPU ACCELERATOR http://www.nvidia.co.jp/content/PDF/kepler/Tesla-K20X-BD-06397-001-v07.pdf
[6] Fujitsu. FUJITSU Supercomputer PRIMEHPC FX100 http://www.fujitsu.com/global/Images/primehpc-fx100-datasheet-en.pdf
[7] G. Latu, M. Haefele, J. Bigot, V. Grandgirard, T. Cartier-Michaud and F.Rozar., Submitted to proceedings of ESAIM, 2015.
[8] Y. Idomura et al, Int. J. HPC. Appl 28, 73-86, 2014.
[9] N. Fujita, et al, The 15th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, 2014.
[10] Marcin Krotkiewski and Marcin Dabrowski, Parallel Computing 39 (10) s 533-548, 2013.
[11] Jianbin Fang, Ana Lucia Varbanescu, Henk J. Sips, Lilun Zhang, Yonggang Che, and Chuanfu Xu, CoRR, 2013.
[12] S. Williams et al., Commun. ACM 52, 65, 2009.