Wilson and Domainwall Kernels on Oakforest-PACS

Issaku Kanamori (Hiroshima Univ.) , Hideo Matsufuru (KEK)

Lattice 2017 at Granada, Spain
June 23, 2017

Motivation and Our Approach

With a new machine/architecture, we need a new code for better performance. We want to know:

- behaviour of the machine
- (minimal) recipe for better performance
- portable code design

we have developed two independent implementations of Wilson/Domainwall kernel:

$\boxed{\text{simple (impl.1)}}$ vs. $\boxed{\text{aggressive (impl.2)}}$

- piling twice more experience
- helps to reveals the machine nature
- how much can we reduce the tuning effort?
- a kind of internal competition

  collaborator(=competitor?) driven code development

Outline

Target Machine: Oarkforest-PACS

- Host: Joint Center for Advanced High Performance Computing
  (U. of Tokyo and Tsukuba U.)
  `http://www.cc.u-tokyo.ac.jp/system/ofp/index-e.html`
- Fastest in Japan (7th in Top 500, June 2017)
  25 PFlops: 3 TFlops [double] $\times$ 8208 nodes
- PRIMERGY CX1640 M1 by Fujitsu:
  Intel Xeon Phi 7250 68C 1.4GHz (KNL) + Intel Omni-Path
  network topology: Full-bisection Fat Tree
  (world largest machine with Omni-Path)
- Full operation since Apr. 2017

Target Machine: KNL

many core, long simd vector

- many core: 2nd generation of Intel Xeon Phi
- 2 core = 1 tile, 1 KNL = 34 tile (7250)
- long simd vector: AVX512
- 16GB MCDRAM: cache, flat, SNC4
  [our benchmark: cache quadrant]
- ...

cf. talk by Harald SERVAT on Tuesday

Development Framework: Bridge++

Bridge++ : Lattice QCD code set [project started 2009, public since 2012]

- C++, object oriented
- intended to be readable, extendable, portable, and with practically high performance
- Framework to develop algorithms, measurements, formulations, etc. by providing common tools
- Testbed for investigation of program design

Development Framework: Extended Bridge++

Bridge++ : Lattice QCD code set [project started 2009, public since 2012]

hot spot tasks are off-loaded to 'alternative code'

- intended for architecture specific codes:
  Many-core, SIMD, vector, accelerators (GPUs, Pezy-SC), etc.
- Previously used for GPUs with OpenCL and OpenACC
  PoS LATTICE2015 (2016) 040; Procedia Computer Science 51 (2015) 1313
- Field as an array with arbitrary index order
- general (portable) code + template classes for a Field array
  specialization of template for architecture specific impl.
- most of the parts are common to architectures (including GPU code)

ideal for developing codes for new machine

Implementation: long SIMD vector in AVX512

long vector variable: 4 complex numbers [double] or 8 [float]

- (clever) compiler + pragma — not so optimal
- low level coding with intrinsics — $\boxed{\text{impl. 1}}$
  - painful to use (for most people), less portable
  - no need of extra libraries
- use of nice library — $\boxed{\text{impl. 2}}$
  - simd directory in Grid P.Boyle et. al PoS LATTICE 2015 (2016) 023
    `https://github.com/paboyle/Grid`
  - easy to use: e.g., $c = a^* b$ bellow

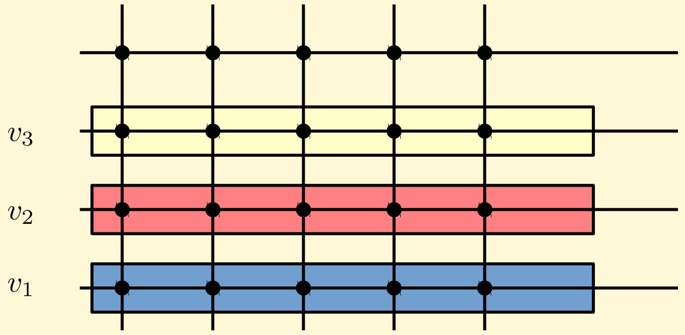
```
// a, b are vector variables
 __m512d a_real = _mm512_shuffle_pd( a, a, 0x00 );
 __m512d a_imag = _mm512_shuffle_pd( a, a, 0xFF );
 a_imag = _mm512_mul_pd( a_imag, _mm512_permute_pd( b, 0x55 ) );
 __m512d c =_mm512_fmsubadd_pd( a_real, b, a_imag );

 // a, b are vComplexD in Grid
 vComplexD c=conj(a)*b;
```

Implementation: some details

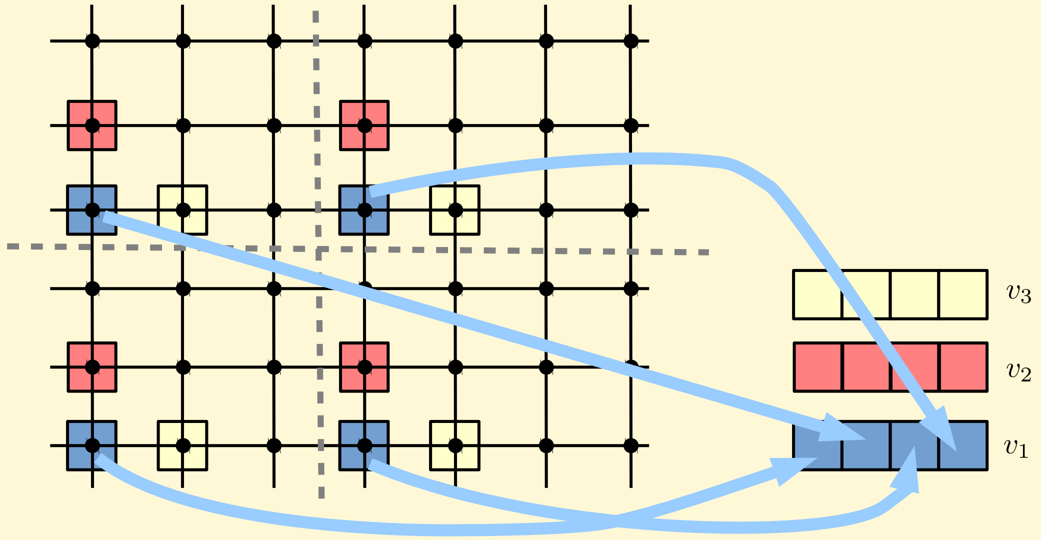
Simple (impl. 1)

- simd vector is continuously packed in x-direction
- no MPI parallelization in x-direction
- blocking comm.
- no manual prefetch
- AVX512 intrinsics for arithmetics



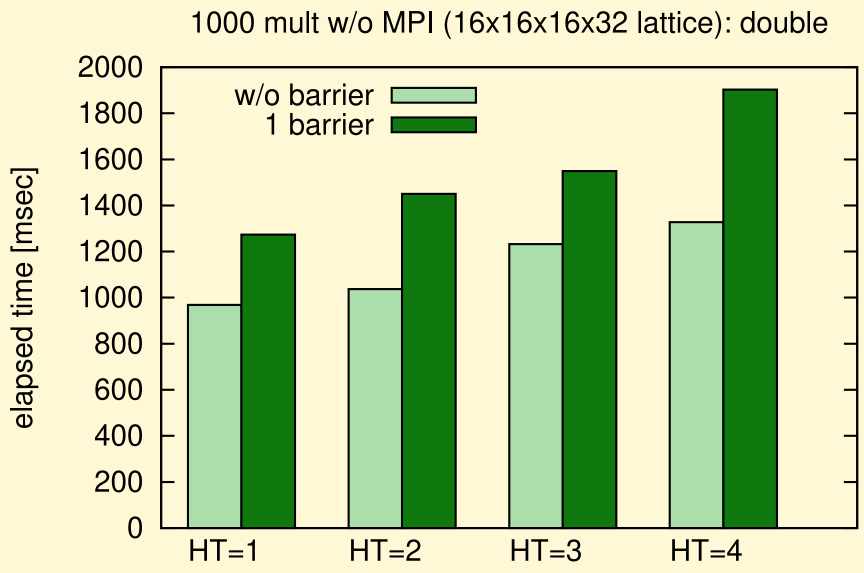
Aggressive (impl. 2)

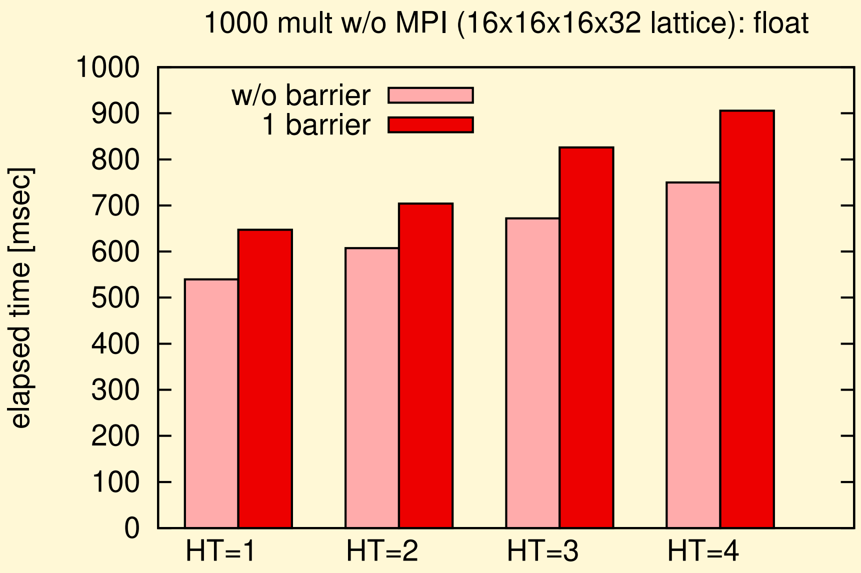
- simd vector is distributed to subdomains (cf. GRID)
- non-blocking comm.
- (partial) loop tiling
- manual prefetch
- `vComplex` from Grid



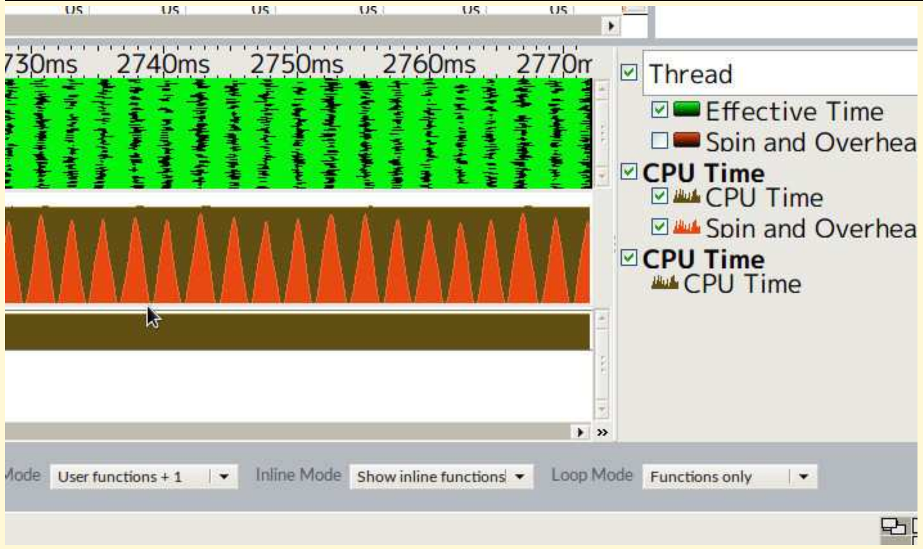
Benchmark: timing and barrier problem

Wilson Kernel for single node (w/o MPI)





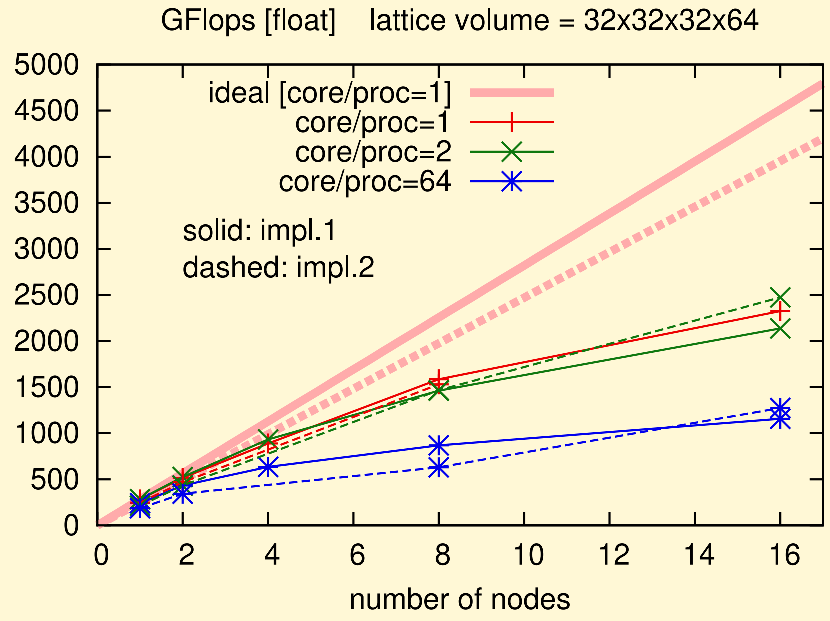
it takes 0.1–0.5 msec / barrier – non-negligible!



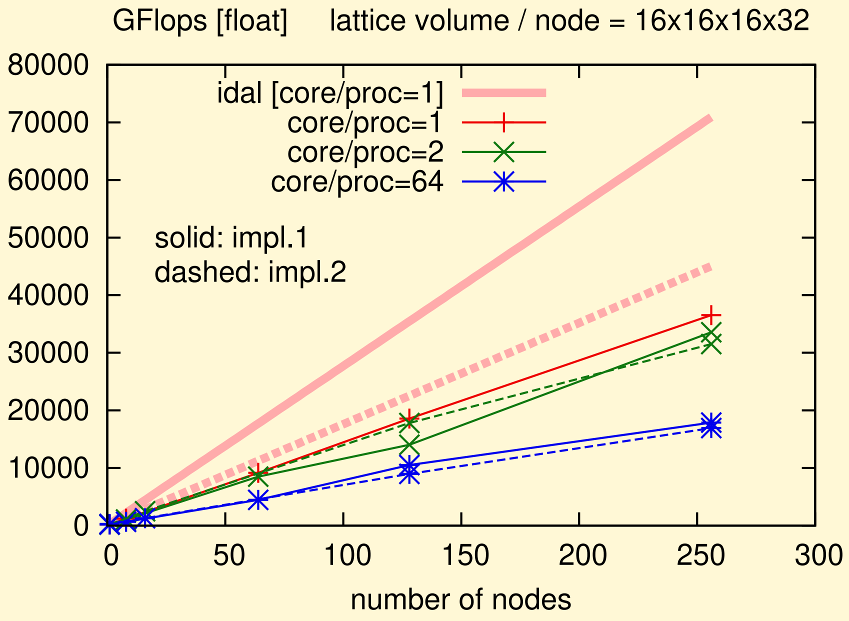
snapshot of Intel Vtune Amplifier: running (green bands) and overhead (red triangles)

Benchmark: Scaling of Wilson Kernel

1 node = 36 tiles, 1 tile = 2 cores



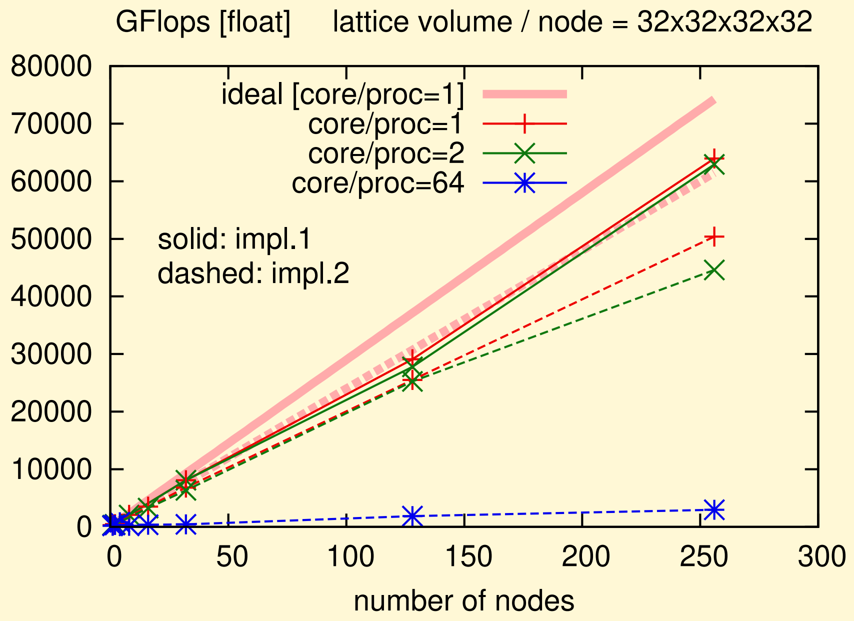
strong scaling



weak scaling

- 32 tiles (64 cores) to the calculations (others to the OS)
- up to 290 GFlops (impl. 1) and 250 GFlops (impl. 2) for # node =1
- roughly 1/2 of the ideal scaling
- fewer cope / MPI processes (= fewer OMP threading) gives better scaling — omp barrier costs a lot

Benchmark: Scaling of Wilson Kernel, cont'd



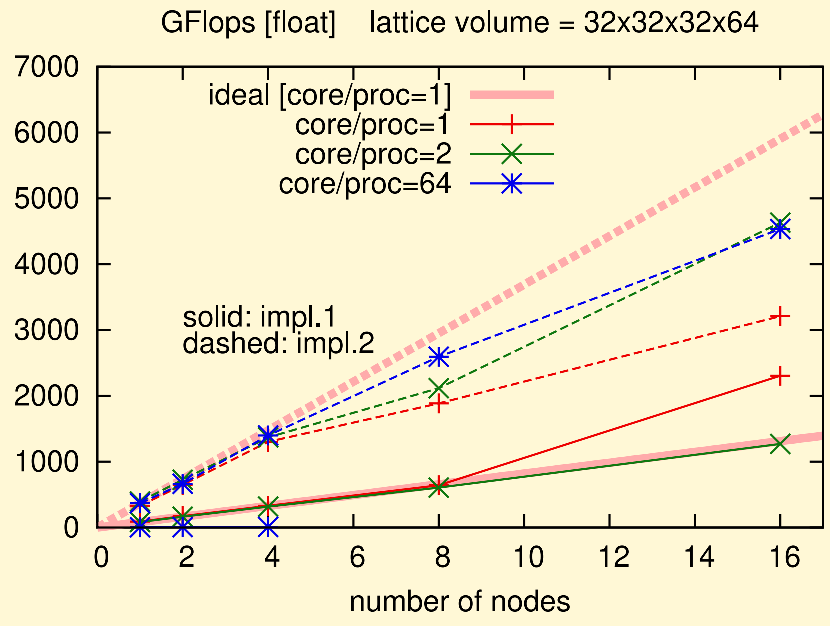
weak scaling

- larger ($16^3 \times 32 \rightarrow 32^4$) lattice volume/ node gives better scaling: 240 GFlops/node (impl. 1)
- 64 cores / MPI process (= 1 proc/node) does not scale MPI? OMP barrier? Affinity? not yet clear

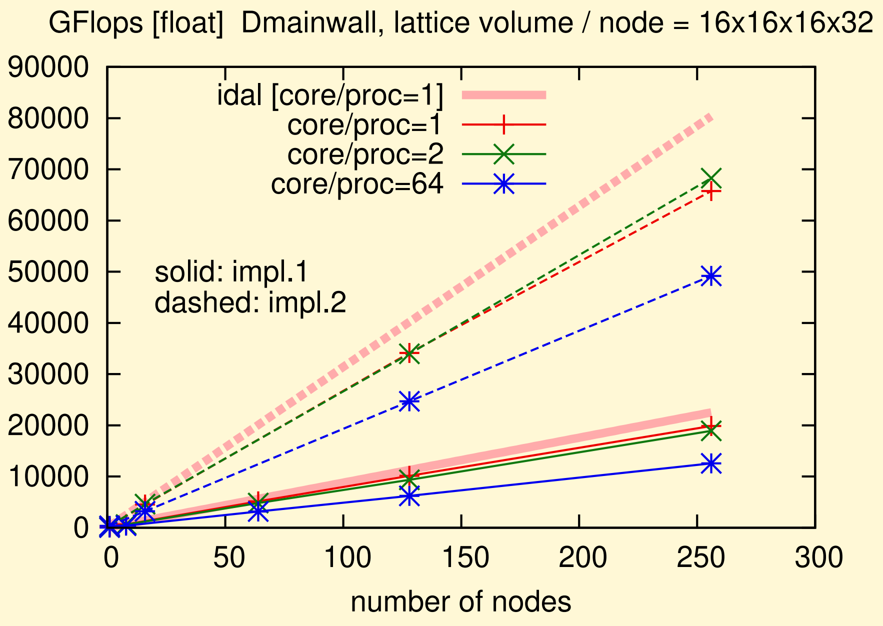
Benchmark: Scaling of Domainwall Kernel

not yet well optimized

1 node = 36 tiles, 1 tile = 2 cores



strong scaling



weak scaling

- 32 tiles (64 cores) to the calculations (others to the OS)
- up to 370 GFlops ($32^3 \times 64$ lat.) and 310 GFlops ($16^3 \times 32$) for # node=1 (impl. 2)
- roughly 80% of the ideal scaling:
  250 GFlops/node for weak, impl. 2
- (non-trivial dep. on the way parallel division)

Conclusions

- code with
  - flexible framework by Bridge++
  - SIMD part: simd directory by Grid (impl. 2)
- performance: not very optimal yet but reasonable
  - “simple” implementation works as well
  - larger volume is better for weak scaling
  - flat MPI (1 MPI/tile or 1 MPI/core) is better
  - OMP barrier seems to be the bottle neck

Conclusions

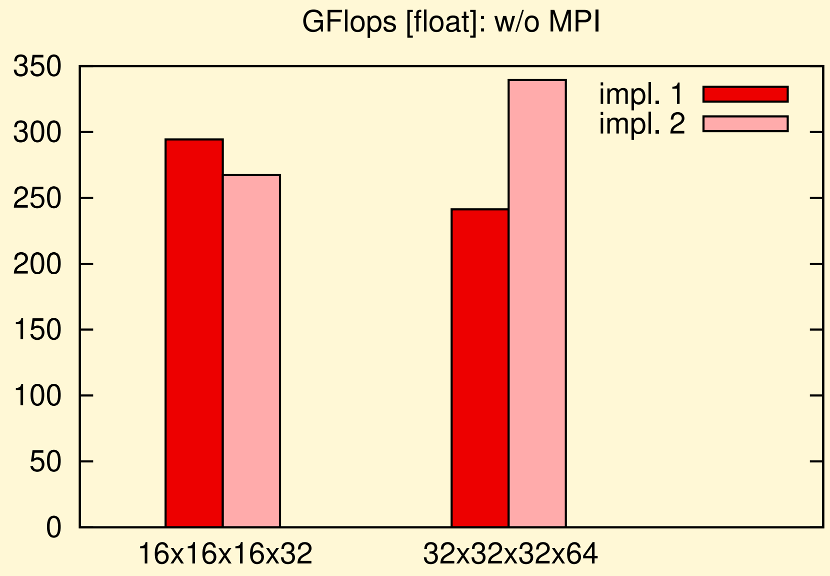
- code with
  - flexible framework by Bridge++
  - SIMD part: simd directory by Grid (impl. 2)
- performance: not very optimal yet but reasonable
  - “simple” implementation works as well
  - larger volume is better for weak scaling
  - flat MPI (1 MPI/tile or 1 MPI/core) is better
  - OMP barrier seems to be the bottle neck

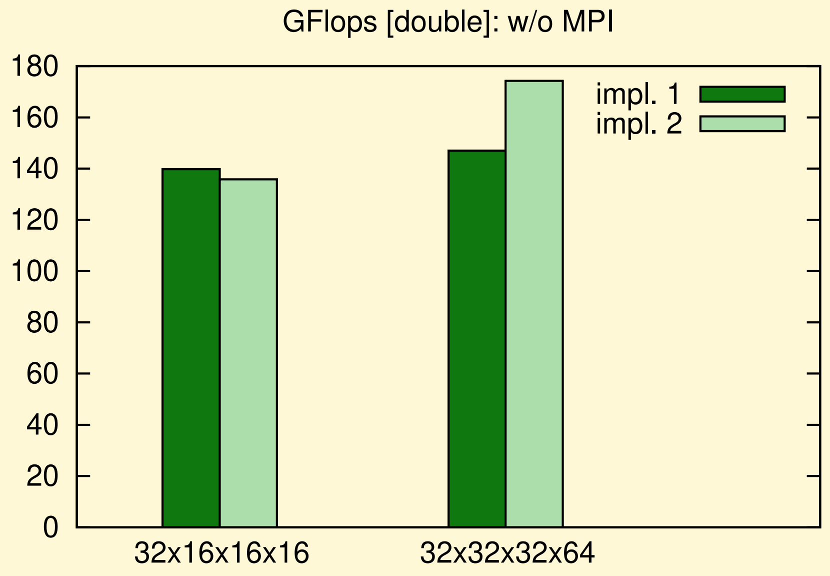
acknowledgements to...

- Recourse (Oarkforest-PACS) was provided by Interdisciplinary Computational Science Program in Center for Computational Sciences, University of Tsukuba
- Priority Issue 9 to be Tackled by Using Post K Computer
- JiCFUS 計算基礎科学連携拠点 Joint Institute for Computational Fundamental Science JICFuS

Backups

Benchmark: in single node, w/o MPI





cf. theoretical peak is 6TFlops [float] and 3TFlops [double]

(around 5–6 % of the peak)