COMPARING RUNTIME SYSTEMS WITH EXASCALE AMBITIONS USING THE PARALLEL RESEARCH KERNELS

Rob Van der Wijngaart¹, Abdullah Kayi¹, Jeff Hammond¹, Gabriele Jost², Tom St. John², Srinivas Sridharan¹, Tim Mattson¹, John Abercrombie³, Jacob Nelson³

¹ Parallel Computing Lab, Intel Corporation
² Data Center Group, Intel Corporation
³ University of Washington
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Extreme Scalability Group Disclaimer

I work in Intel Labs and therefore don't know anything about Intel products.

I am not an official spokesman for Intel.

I do not speak for my collaborators, whether they be inside or outside Intel.

You may or may not be able to reproduce any performance numbers I report.

Hanlon’s Razor (blame stupidity, not malice).
HPC software design challenges

- To MPI or not to MPI...
- One-sided vs. two-sided?
- Does your MPI/PGAS need a +X?
- Static vs. dynamic execution model?
- What synchronization motifs maximize performance across scales?

Application programmers can afford to rewrite/redesign applications zero to one times every 20 years...
Programming model evaluation

Standard methods

• NAS Parallel Benchmarks
• Mini Applications (e.g. Mantevo, LULESH)
• HPC Challenge

There are numerous examples of these on record, covering a wide range of programming models, but is source available and curated*?

What is measured?

• Productivity (?), elegance (?)
• Implementation quality (runtime or application)
• Asynchrony/overlap
• Semantics:
  • Automatic load-balancing (AMR)
  • Atomics (GUPS)
  • Two-sided vs. one-sided, collectives

* PRK curation is currently active. - no future commitment, but you can GitHub fork and carry the torch.
Goals of the Parallel Research Kernels

1. **Universality**: Cover broad range of performance critical application patterns.

2. **Simplicity**: Concise pencil-and-paper definition and transparent C reference implementation. *No domain knowledge required.*

3. **Portability**: Should be implementable in any sufficiently general programming model.

4. **Extensibility**: Parameterized to run at any scale. Other knobs to adjust problem or algorithm included.

5. **Verifiability**: Automated correctness checking and built-in performance metric evaluation.

6. **Hardware benchmark**: No! Use HPCChallenge, Xyz500, etc. for this.
Outline of PRK Suite

- **Dense matrix transpose**
- Synchronization: global
- **Synchronization: point to point**
- Scaled vector addition
- Atomic reference counting
- Vector reduction
- Sparse matrix-vector multiplication
- Random access update
- **Stencil computation**
- Dense matrix-matrix multiplication
- Branch
- Particle-in-cell (new)
- (more in progress)

\[ A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1} \]
PRK implementations

- Serial (C89)
- OpenMP (C89, C99)
- MPI1 – MPI two-sided
  - Also support FG-MPI (UBC) and AMPI (UIUC)
- MPIOMP – MPI two-sided with local OpenMP
- MPISHM – MPI two-sided with MPI-3 shared-memory
- MPIRMA – MPI one-sided communication (multiple flavors)
- SHMEM (C89)
- UPC
- Grappa (C++)
- Charm++ (C++)
- Fortran 2008 (serial, OpenMP, coarrays, intrinsics)
- Python (simple and Numpy)
- Julia

Serial C, OpenMP, and MPI support most of the PRKs. **Synch_p2p, Stencil** and **Transpose** are primary targets for distributed-memory evaluation.

In progress:
- Legion (Stanford)
- HPX (LSU & IU)
- OCR (Rice/Intel)
- Chapel (Cray)
(Kernel, Implementation) matrix is not full rank...

- Synch_p2p, Stencil, and Transpose supported by all models.
- Serial, OpenMP and MPI1 (incl. AMPI and FG-MPI) support all static kernels.
- PIC only supported by Serial and MPI1 (this will change).
- Chapel, Fortran 2008 and Python support multiple styles:
  - Fortran: serial+OpenMP, coarrays, pretty (intrinsics or array notation)
  - Python: basic and Numpy (Stencil and Transpose)
- MPI RMA uses different idioms for different kernels – fill-in desired.
- C99 VLA with serial and OpenMP not yet merged.
This is a set of simple programs that can be used to explore the features of a parallel platform.
https://groups.google.com/forum/#!forum/parallel-research-kernels

1,252 commits
11 branches
5 releases
10 contributors

- AMPI: more missing -lm
- CHARM++: more missing -lm
- FG_MPI: add more missing libm for sqrt and ceil
- FORTRAN: Merge pull request #106 from jeffhammond/julia
- GRAPPA: Adding debug info to GRAPPA((Stencil,Transpose)
- JULIA: remove temp files [ci skip]
- LEGION: Adding OpenMP version of PIC, reverting OpenMP Transpose to "old" ver...
- MPI1: more missing -lm
- MPIOPENMP: add more missing libm for sqrt and ceil
- MPRIRMA: sqrt() require libm - resolves #102

Latest commit 54da8be 12 days ago

Branch: master → New pull request

Find file Clone or download →
for i in range(1,m):
    for j in range(1,n):
        grid[i][j] = grid[i-1][j] + grid[i][j-1] - grid[i-1][j-1]

grid[0][0] = -grid[m-1][n-1]

A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}

- Proxy for discrete ordinates neutron transport (e.g. PARTSN)
- Much simpler than SNAP proxy.
- 1D decomposition for distributed memory.
- 1-word messages from (to) the left (right) neighbors for each row.
- Wraparounds to create dependency.
+ W[0,2] * A[0:n-4,2:n-2]
+ W[1,2] * A[1:n-3,2:n-2]

- Proxy for structured mesh codes. 2D stencil to emphasize non-compute.
- Supports arbitrary radius star and square stencils.
- 2D decomposition with ghost cells for distributed-memory neighbors.
- Dim_x/npoc_x-word messages for exchange.
for i in range(order):
    for j in range(order):
        B[i][j] += A[j][i]
        A[j][i] += 1.0

• Proxy for 3D FFT, bucket sort...
• 1D decomposition for distributed-memory; blocking for other dimension to avoid doubling storage (i.e. 2D tiling of compute).
• Local transpose of square tiles supports blocking to reduce TLB pressure.
• SHMEM is put-driven, coarrays is get-driven...
Experimental apparatus

All experiments strong-scale 49152x49152 (47104x47104 for Charm++)
Details

- Use Intel compilers by default (i.e. we eat our own dog food).
- Cray UPC uses the Cray C compiler.
- Grappa uses GCC because Intel cannot (yet) be used (i.e. bug).
- Compiler differences found to be small in limited investigation.
- Use best available communication libraries by default.
  - Berkeley UPC uses GASNet over uGNI; Charm++ uses uGNI.
  - FG-MPI not available except TCP/IP, so cannot evaluate (properly) on Cray.

http://www.nersc.gov/users/computational-systems/edison/configuration/
Normalized performance (to MPI1 node)

- MPI1
- MPISHM12
- MPISHM24
- MPIOPENMP
- MPRMA
- SHMEM
- BUPC
- CRAYUPC
- CHARM++1
- CHARM++4
- CHARM++16
- GRAPPA

Cores:
- 24
- 48
- 96
- 192
- 384
- 768
- 1536
- 3072
- 6144
- 12288
msg size (B) = 
\[8 \times (49152 / \text{nproc})^2\]

e.g.

\[8 \times (49152 / 1536)^2 = 8192\]

\[8 \times (49152 / 12288)^2 = 128\]

\[8 \times (49152 / 512)^2 = 73728\]
msg size (B) = 8*(49152/nproc)^2

- e.g.
  - 8*(49152/3072)^2 = 2048
  - 8*(49152/6144)^2 = 512
  - 8*(49152/12288)^2 = 128
Summary

• PRK designed to expose semantic and implementation differences between programming models for important HPC application patterns.

• Ported 3 PRKs to 7+ models in one year with relatively modest effort. Nontrivial porting efforts associated with learning new programming model, not the PRKs themselves.

• First-generation PRKs are relatively static. Actively developing dynamic PRKs. Asynchronous programming models expected to do better here.

• Root cause analysis of performance results treated as out-of-scope. (Not having e.g. MPI source code makes this difficult.)
References


jhammond@nid00081:~/PRK/git> srun -c 1 -N 32 -n $(32*24) ./MPIRMA/Transpose/transpose 10 $((32*24*2)) 32
Parallel Research Kernels version 2.16
MPIRMA matrix transpose: B = A^T
Number of ranks = 768
Matrix order = 1536
Number of iterations = 10
Tile size = 32
Synchronization = MPI_Win_fence
Block_size = 4
Solution validates
Rate (MB/s): 1348.294420 Avg time (s): 0.027997

jhammond@nid00081:~/PRK/git> export MPICH_RMA_OVER_DMAPP=1

jhammond@nid00081:~/PRK/git> srun -c 1 -N 32 -n $(32*24) ./MPIRMA/Transpose/transpose 10 $((32*24*2)) 32
Parallel Research Kernels version 2.16
MPIRMA matrix transpose: B = A^T
Number of ranks = 768
Matrix order = 1536
Number of iterations = 10
Tile size = 32
Synchronization = MPI_Win_fence
Block_size = 4
Solution validates
Rate (MB/s): 42064.206801 Avg time (s): 0.000897
Parallel Research Kernels version 2.16

MPIRMA matrix transpose: B = A^T

Number of ranks = 1536
Matrix order = 3072
Number of iterations = 10
Tile size = 32
Synchronization = MPI_Win_fence
Block size = 4
Solution validates
Rate (MB/s): 84472.903258 Avg time (s): 0.001787

Number of ranks = 1536
Matrix order = 6144
Number of iterations = 10
Tile size = 32
Block size = 16
Rate (MB/s): 64184.236420 Avg time (s): 0.009410

Number of ranks = 1536
Matrix order = 12288
Number of iterations = 10
Tile size = 32
Block size = 64
Rate (MB/s): 134059.110342 Avg time (s): 0.018021
The PRK community

April 10, 2016 – May 10, 2016

Overview

1 Active Pull Request
1 Merged Pull Request
0 Proposed Pull Requests

2 Active Issues
2 Closed Issues
0 New Issues

Excluding merges, 5 authors have pushed 56 commits to master and 66 commits to all branches. On master, 16 files have changed and there have been 2,212 additions and 10 deletions.

1 Pull request merged by 1 person
#95 Legion implementation of Stencil and Transpose 8 days ago

2 Issues closed by 1 person
#84 Fortran 2008 pretty stencil 7 days ago
#96 Legion breaks 'make clean' 7 days ago

This does not include email contributions...
Synch_p2p, strong scaled (49152x49152*)

Aggregate performance MFlops
Stencil, strong scaled (49152x49152*)

Normalized performance (Mflops/#nodes)/Mflops_single_node_MPI1
Transpose, strong scaled (49152x49152*)

MPI+X based models win (X=OpenMP/MPI3)

Aggregate performance MB/s

* Charm++: (47104x47104)