SAND2019-0208C

Proxy Apps Mysteries Revealed

PROXY APPLICATIONS Co-Design

PI: David Richards (LLNL)
Team: Jeanine Cook (SNL), Hal Finkel (ANL), Christoph Junghans (LANL), Peter McCorquodale (LBNL) Shirley Moore (ORNL), Omar Aaziz (SNL), Tanner Juedeman (SNL), Courtenay Vaughan (SNL), Brian Homerding (ANL), Thomas Uram (ANL), Abhinav Bhatele (LLNL), Xavier Andrade (LLNL), Robert Pavel (LANL), Vinay Ramakrishnaiah (LANL), Tiffany Mintz (ORNL), Greg Watson (ORNL)

Mission

To curate a suite of proxy applications that are representative of the intended characteristics of their respective parent applications and are easy to obtain and use. Characteristics include hardware bottlenecks (e.g., memory, computation, communication) and programming models.

Proxy App Suite v2.0

- 53 proxy applications now in our catalog, below are the top 15:

AMG	miniVite
CANDLE Benchmarks	NEKbone
Ember	PICSARlite
ExaMiniMD	SW4lite
Laghos	SWFFT
MACSio	thornado-mini
miniAMR	XSBench
miniQMC	

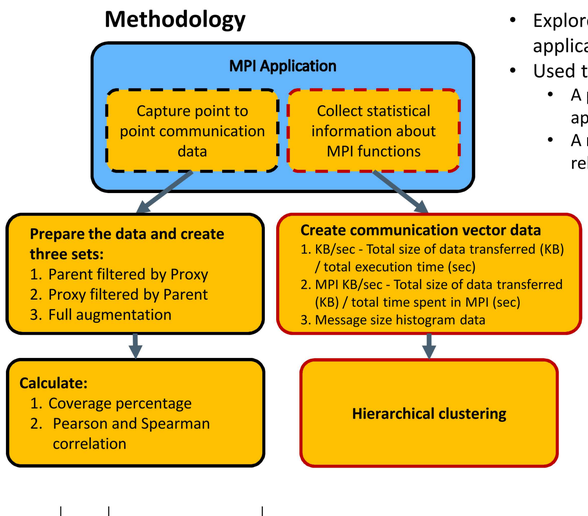
- Each proxy app can be installed with a simple Spack command
  - Dependencies are easily identified and provided
  - See https://proxyapps.exascaleproject.org/downloads
- Improved metadata to track provenance and sponsors of proxies
- The Proxy App Team worked with ECP Projects to develop or enhance new proxy apps
  - PICSARlite: Reduced code size by 80%
  - miniVite: Consulted on design of proxy and selection of algorithm
  - thornado-mini: Helped with documentation and testing

Quantitative Assessment

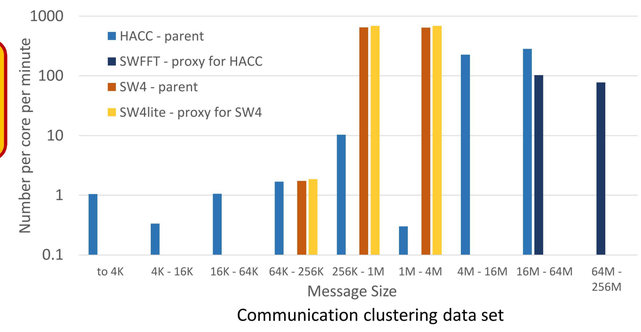
Goal: Understand how well proxies represent parent applications quantitatively at hardware level ➔ better proxies in future

- Representative problems/sizes
- Detailed profiling
- Quantitative characterization
- Statistical proxy/parent app comparison

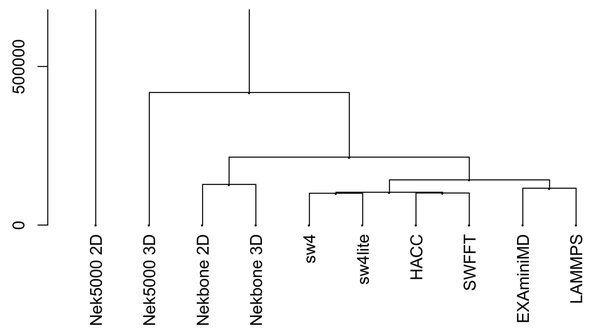
Quantitative Comparison



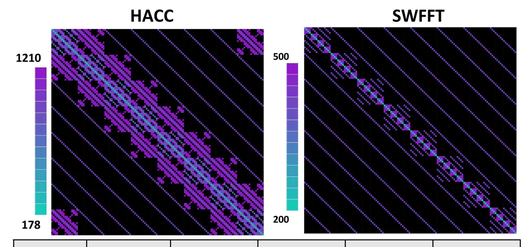
- Explored how the communication of a proxy application relates to its parent application
- Used two quantification methods:
  - A pairwise communication quantification method that captures how much of one application matches the other
  - A message characteristics quantification method that produces a clustering-based relatedness measures of the parent applications and their proxies



Communication clustering data set



Communication Similarity, Clustering Message Size and Frequency



Parent/Proxy	Parent in Proxy		Proxy in Parent		Full Set		Parent in Proxy		Proxy in Parent	
	#msg	#pair	#msg	#pair	PCorr	SCorr	PCorr	SCorr	PCorr	SCorr
HACC/ SWFFT	68.7	41.1	100	100	0.97	0.99	0.92	0.84	0.97	0.99

Point to Point Communication Analysis

Proxy Catalog Performance Characterization

Latency

HPCCG

A simple conjugate gradient benchmark code for a 3D chimney domain on an arbitrary number of processors.

Problem Size Discussion

From the application README

Suggested: Data size is over a range from 25% of total system memory up to 75%.

With `nxnynz` and `n = nx * ny * nz`
Total memory per MPI rank: `720 * n` bytes for 27 pt stencil, `240 * n` bytes for 7 pt stencil.

Additional details in application README

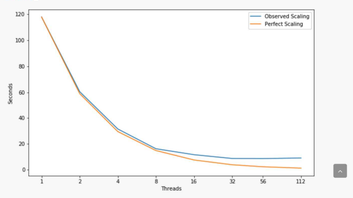
Analysis

On the **Skylake** machine on which it was analyzed, **HPCCG** uses **70-80% of the DRAM bandwidth** despite having **memory latency** issues due to a **indirect memory access**.

Parameters

```
Compiler = icc (ICC) 18.0.1 20171018
Build_Flags = -g -O3 -march=native -ftree-vectorize -qopenmp -DUSING_OMP
Run_Parameters = 256 256 256
```

Scaling



`HPC_sparsemv.cpp`

Threads (Time)	IPC per Core	Loads per Cycle	L1 Hits per Cycle	L1 Miss Ratio	L2 Miss Ratio	L3 Miss Ratio	L2 B/W Utilized	L3 B/W Utilized	DRAM B/W Utilized
1 (83.1%)	1.39	0.77	0.68	1.86%	42.24%	72.59%	11.36%	38.16%	42.23%
56 (70.4%)	0.47	0.25	0.22	1.74%	42.51%	68.42%	3.27%	28.55%	78.07%
112 (65.5%)	0.48	0.13	0.12	2.19%	42.56%	75.15%	3.51%	28.05%	76.01%

```
66 int HPC_sparsemv( HPC_Sparse_Matrix *A,
67 const double * const x, double * const y)
68 {
69
70   const int nrow = (const int) A->local_nrow;
71
72   #ifdef USING_OMP
73   #pragma omp parallel for
74   #endif
75   for (int i=0; i< nrow; i++)
76   {
77     double sum = 0.0;
78     const double * const cur_vals =
79     (const double * const) A->ptr_to_vals_in_row[i];
80
81     const int * const cur_inds =
82     (const int * const) A->ptr_to_inds_in_row[i];
83
84     const int cur_nnz = (const int) A->nnz_in_row[i];
85
```

Threads (Time)	IPC per Core	Loads per Cycle	L1 Hits per Cycle	L1 Miss Ratio	L2 Miss Ratio	L3 Miss Ratio	L2 B/W Utilized	L3 B/W Utilized	DRAM B/W Utilized
1 (64.5%)	1.32	0.62	0.55	1.92%	43.27%	71.10%	10.02%	37.62%	39.92%
56 (63.7%)	0.41	0.20	0.17	1.78%	44.29%	63.23%	2.87%	28.25%	75.00%
112 (60.1%)	0.47	0.13	0.11	2.20%	43.91%	74.23%	3.28%	27.66%	73.94%

```
86     for (int j=0; j< cur_nnz; j++)
87       sum += cur_vals[j]*x[cur_inds[j]];
88     y[i] = sum;
89   }
90   return(0);
91 }
```

Bandwidth

CloverLeaf

From http://uk-mac.github.io/CloverLeaf/

CloverLeaf is a mini-app that solves the compressible Euler equations on a Cartesian grid, using an explicit, second-order accurate method. Each cell stores three values; energy, density, and pressure. A velocity vector is stored at each cell corner. This arrangement of data, with some quantities at cell centers, and others at cell corners is known as a staggered grid.

The computation in CloverLeaf has been broken down into "kernels" — low level building blocks with minimal complexity. Each kernel loops over the entire grid and updates one (or some) mesh variables, based on a kernel-dependent computational stencil. Control logic within each kernel is kept to a minimum , allowing maximum optimisation by the compiler. Memory is sacrificed in order to increase peformance, and any updates to variables that would introduce dependencies between loop iterations are written into copies of the mesh.

Problem Size Discussion

From README.md

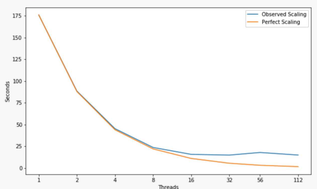
There are four standard input files that are recommended for testing:

1. `clover_bm_short.in` – This is not a very sensitive test and the kinetic energy at the end of this run should be 0.1193E+01.
2. `clover_bm.in` – This runs for 2955 timesteps and is more sensitive than the first test. Through this simulation the whole computational mesh in traversed by a shock and so it is a good test of the parallel implementation because all internal boundaries will be crossed during the course of the simulation. The final kinetic energy should be 0.2590E+01.
3. `clover_bm16_short.in` – This is the "socket" test and has a much larger mesh size and therefore, memory footprint. The final kinetic energy should be 0.3075E+00.
4. `clover_bm16.in` This is a fairly long, large mesh run and the kinetic energy at the final time should be 0.4854E+01

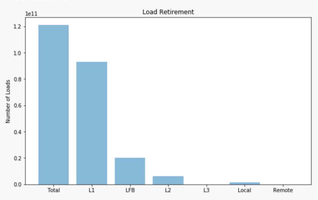
Analysis

CloverLeaf becomes bound by the **DRAM bandwidth**, partly resulting from high **cache miss** rates on the **Skylake** machine which this analysis was performed.

Scaling



Hit Locations



FLOPS

Double Precision	Scalar	128B Packed	256B Packed	512B Packed	Total FLOPS	GFLOPS/sec
PMU	3.220e+08	2.510e+10	1.390e+11	0.000e+00	6.065e+11	3.866e+01
SDE	3.702e+08	3.145e+10	1.297e+11	0.000e+00	5.822e+11	3.711e+01

Compute

RSBench

From README.txt

RSBench is mini-app to represent the multipole resonance representation lookup cross section algorithm for Monte Carlo neutron transport.

Problem Size Discussion

From README.txt

Problem size should be scaled using the `-p <particles>` parameter. The default is set to 300,000, but full application runs may use up to several billion particles per generation.

Analysis

RSBench was **compute bound** on the **Skylake** machine that this analysis was run on and spent 27.1% of its time in **libm**.

FLOPS

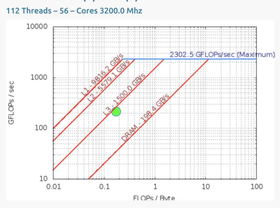
Double Precision	Scalar	128B Packed	256B Packed	512B Packed	Total FLOPS	GFLOPS/sec
PMU	5.090e+11	1.970e+12	1.780e+11	0.000e+00	5.161e+12	2.157e+02
SDE	2.898e+11	1.004e+12	8.096e+10	0.000e+00	2.621e+12	1.095e+02

Intel Software Development Emulator

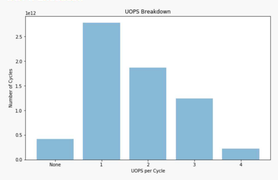
Intel SDE	RSBench
Arithmetic Intensity	0.175
FLOPS per Inst	0.83
FLOPS per FP Inst	1.91
Bytes per Load Inst	12.1
Bytes per Store Inst	13.0

Roofline – Intel(R) Xeon(R) Platinum 8180M CPU

112 Threads – 56 – Cores 3200.0 Mhz



UOPS Executed

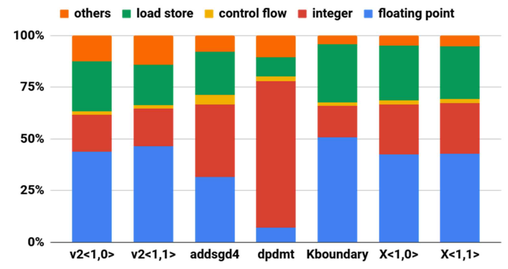


Added additional information to the proxy app catalog

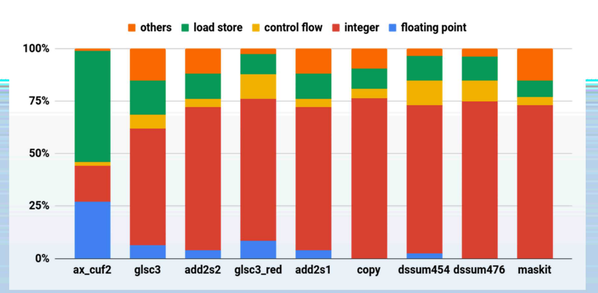
- Performance bounds and data
- Problem size and run information
- Important code segments with cache and memory bandwidth data

GPU Characterization of Proxy Apps

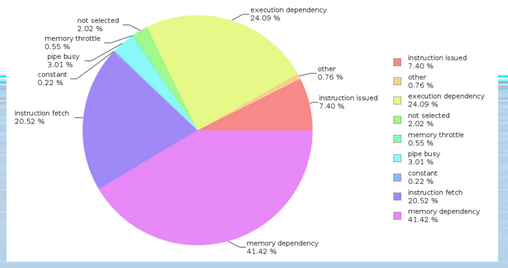
Instruction mix for PENNANT (sedovbig) and Quicksilver



Instruction mix for SW4lite (LOH1-h100)



Instruction mix for NEKbone



Stall cycle breakdown for PENNANT gpumain2 kernel on V100

Observations:

- All four proxy apps are limited by memory performance on P100 and V100 GPUs.
- Applications that are latency bound on P100 remain latency bound to same degree on V100, in spite of improvements to cache/memory hierarchy.
  - SW4lite improved from 14% to 16% of peak
  - PENNANT remained at 8%
- Above instruction mixes are shown for P100. Integer instruction counts, which include address calculations, are reduced and can be performed simultaneously with floating point operations on V100 and thus have less impact on V100.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
SAND No. _____