Characterizing and Predicting Scientific Workloads for Heterogeneous Computing Systems

Beau Johnston

Panel: Josh Milthorpe, Greg Falzon and Alistair Rendell

The Australian National University

29 January, 2019

Supercomputers 101

- Used in computationally intensive tasks and are a critical component in current scientific research.
- Essential in simulations for quantum mechanics, weather forecasting, climate research, oil and gas exploration and molecular modeling, etc.
- However require huge amounts of electricity to operate – the Summit, currently number one in the TOP500, requires 8.8 MW to power, which is in the capacity of a small coal power plant.
- To reduce this large energy footprint supercomputers are becoming increasingly heterogeneous.

Trends in Supercomputing – A view from the Top500

1. Summit – GV100
2. Sunway TaihuLight
3. Sierra – GV100
4. Tianhe-2A
5. ABCI – V100
6. Piz Daint – P100
7. Titan – K20x
8. Sequoia
9. Trinity – Phi
10. Cori – Phi

Trends in Supercomputing – A view from the Top500

1. Summit – GV100
2. Sunway TaihuLight
3. Sierra – GV100
4. Tianhe-2A
5. ABCI – V100
6. Piz Daint – P100
7. Titan – K20x
8. Sequoia
9. Trinity – Phi
10. Cori – Phi

- As of June 2018
- 7 / 10 use accelerators
- Newest is Summit – based on IBM Power9 with NVLINK and CAPI
- All devices have an OpenCL runtime
- Reliance on accelerators is increasing
- Scheduling to the most appropriate device is an open problem

Trends in Supercomputing – Accelerator Use

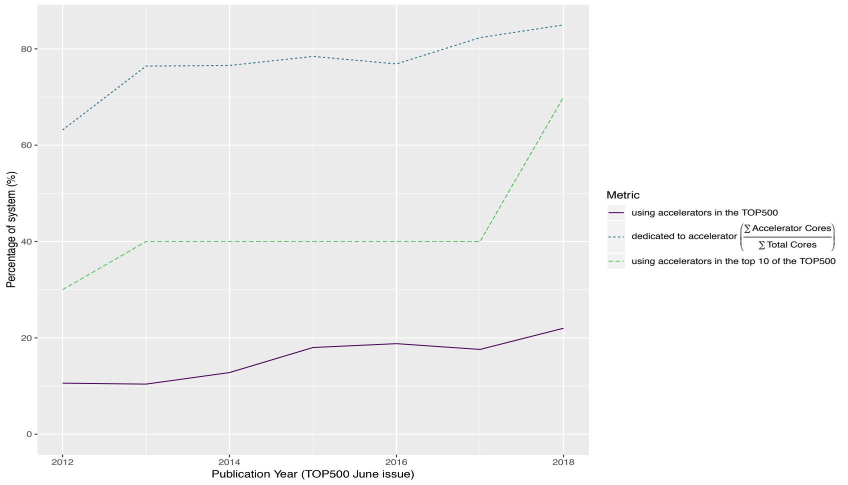


Figure 1: The percentage of accelerators in use and the contributions of cores found on systems with accelerators in the Top500.

Trends in Supercomputing – Accelerator Energy

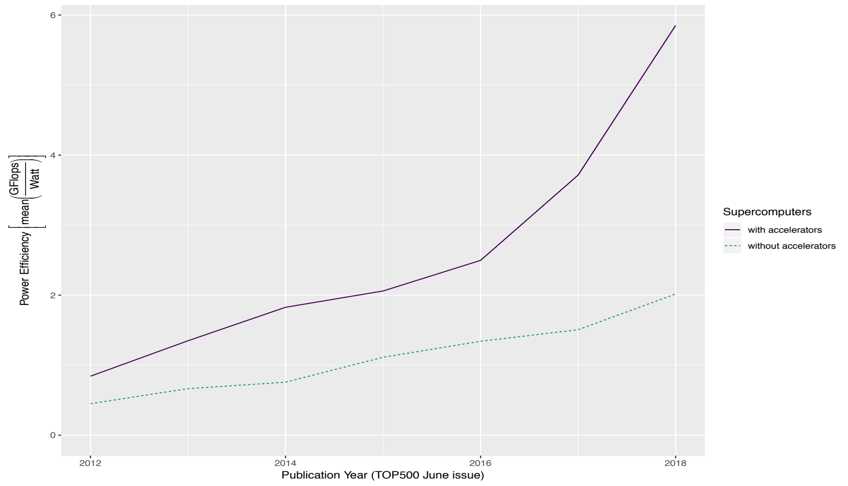


Figure 2: Power efficiency (GFlops/Watt) of using accelerators in the Top500 supercomputers over time.

Schedulers and the Importance of Prediction

- Increasingly heterogeneous at a node level.
- Schedulers responsible for choosing the right tool for the job.
- Accelerator aware schedulers include StarPU [1], Ompss [2] and CoreTSAR [3].

[1] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," Concurrency and Computation: Practice and Experience, vol. 23, no. 2, pp. 187–198, 2011.

[2] A. Duran et al., "Ompss: A proposal for programming heterogeneous multi-core architectures," Parallel Processing Letters, vol. 21, no. 02, pp. 173–193, 2011.

[3] T. R. Scogland, W.-c. Feng, B. Rountree, and B. R. de Supinski, "Coretsar: Adaptive worksharing for heterogeneous systems," in International supercomputing conference, 2014, pp. 172–186.

The Problem

- Schedulers track dependencies within tasks and schedule to minimize, compute time/energy, compute bandwidth and latency.
- Scheduling work to the most appropriate accelerator at the granularity of function call level or the work inside a single parallel region.
- Better manage resources of supercomputers by keeping codes running on the best accelerator.
- *But* current approaches run every new code on all available accelerators to determine initial performance → wasteful!

Thesis Statement

“Scientific high performance computer systems are becoming increasingly heterogeneous because certain applications are more suitable to particular accelerators. The characteristics of a program are sufficient to predict its performance which will allow the efficient scheduling of work to the most appropriate accelerator.”

Talk Structure

1. Benchmarking & Gauging the Performance of HPC Accelerators
2. Workload Characterization
3. Making Predictions
4. Use in the Scheduler Setting
5. Conclusions and Future Work

OpenCL – The Language of Heterogeneous Computing

- Open Computing Language (OpenCL) is an open standard.
- Allows computationally intensive codes – kernels – to be written once and run on any compliant accelerator.
- Most vendors are compliant to basic standards.
- Application code can be written directly in OpenCL, and
- Can be used as a back-end for higher level languages – OpenMP runtime implemented for TI Keystone II DSP architecture[4].
- Increased relevancy for FPGA programming

[4] Mitra, G. et al. 2014. Implementation and optimization of the OpenMP accelerator model for the TI Keystone II architecture. International workshop on openmp (2014), 202–214.

Dwarfs and the Diversity of Scientific Computing

- Large pool of applications but how to know they are diverse and compare devices under a real-world scientific load?
- Instead of traditional benchmarks, use 13 “Dwarfs” to design and evaluate parallel programming models and architectures.[5]
- A dwarf is an algorithmic method that captures a pattern of computation and communication.
- e.g., Dense Linear Algebra applications generally use unit-stride memory accesses to read data from rows and strided accesses to read data from columns, while Map Reduce calculations depend on statistical results of repeated random trials and are considered embarrassingly parallel.

[5] K. Asanovic et al., "The landscape of parallel computing research: A view from Berkeley," EECS Department, University of California, Berkeley, UCB/EECS-2006-183, 2006.

Benchmarking

- Examine the performance characteristics of scientific codes – and the suitability of different accelerators.
- 3 OpenCL suites were considered:

  1. SHOC – focused on micro-kernels rather than complete applications
  2. Rodinia – targets language comparison and not all dwarfs covered
  3. **OpenDwarfs** – selected since it offers the widest range of benchmarks covering all dwarfs

Extended Open Dwarfs Benchmark Suite

- Extended Open Dwarfs (EOD) Benchmark Suite
- Based off the OpenDwarfs benchmark suite[6]
- Benchmarks selected following diversity analysis and 13 Berkeley Dwarfs taxonomy
- Built in OpenCL
- Purpose of OpenDwarfs was a to characterize a diverse set of parallel applications and architectures using a common language, but had deficiencies. . .

[6] Krommydas, K. OpenDwarfs: Characterization of dwarf-based benchmarks on fixed and reconfigurable architectures. Journal of Signal Processing Systems, vol. 85, no. 3, pp. 373-392, 2016

OpenDwarfs Deficiencies

- Architecture specific optimizations
- Hard-coded device specific parameters
- Fixed problem sizes

EOD Extensions

- Selection of problem size critically affects HPC benchmarking
- Highest impact on CPU architectures.
- A major contribution of the work is facilitating 4 different problem sizes for all applications presented in the suite.
- Selected according to levels of cache
  - **tiny** : $<$ 32 KiB L1
  - **small**: $<$ 256 KiB L2
  - **medium**: $<$ 8192 L3
  - **large**: $>$ 8192 L3

EOD Extensions II

- Diverse:
  - 4 different problem sizes per application
  - Added `dwt` and `fft` applications – currently 11 and 37 kernels
  - Real scientific applications
- Reproducible: Minimum of 2 sec runs per benchmark
- Precise:
  - High-resolution timers with LibSciBench[7]
  - Reported with one cycle resolution and roughly 6 ns of overhead
  - Also allows collection of energy and hardware events

[7] T. Hoefler and R Belli. "Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015. ACM, 73.

EOD Extensions III

- Portable:
  - Based on an OpenCL backend
  - Tested on a wide range of hardware
  - Consistent tuning – i.e. workgroup size arguments

Applications

Table 1: List of Extended OpenDwarfs Applications and their respective dwarfs

Dwarf	Extended OpenDwarfs Application
Dense Linear Algebra	LU Decomposition
Sparse Linear Algebra	Compressed Sparse Row
Spectral Methods	DWT2D, FFT
N-Body Methods	Gemnoui
Structured Grid	Speckle Reducing Anisotropic Diffusion
Unstructured Grid	Computational Fluid Dynamics
Map Reduce	K-Means
Combinational Logic	Cyclic-Redundancy Check
Graph Traversal	Breadth First Search
Dynamic Programming	Smith-Waterman
Backtrack and Branch and Bound	N-Queens
Graphical Methods	Hidden Markov Models
Finite State Machines	Temporal Data Mining

Experimental Setup

Table 2: Hardware used for the EOD evaluation.

Name	Vendor	Type	Series	Core Count	Clock Frequency (MHz) (min/max/turbo)	Cache (KiB) (L1/L2/L3)	TDP (W)	Launch Date
Xeon E5-2697 v2	Intel	CPU	Ivy Bridge	24∗	1200/2700/3500	32/256/30720	130	Q3 2013
i7-6700K	Intel	CPU	Skylake	8∗	800/4000/4300	32/256/8192	91	Q3 2015
i5-3550	Intel	CPU	Ivy Bridge	4∗	1600/3380/3700	32/256/6144	77	Q2 2012
Titan X	Nvidia	GPU	Pascal	3584†	1417/1531/–	48/2048/–	250	Q3 2016
GTX 1080	Nvidia	GPU	Pascal	2560†	1607/1733/–	48/2048/–	180	Q2 2016
GTX 1080 Ti	Nvidia	GPU	Pascal	3584†	1480/1582/–	48/2048/–	250	Q1 2017
K20m	Nvidia	GPU	Kepler	2496†	706/–/–	64/1536/–	225	Q4 2012
K40m	Nvidia	GPU	Kepler	2880†	745/875/–	64/1536/–	235	Q4 2013
FirePro S9150	AMD	GPU	Hawaii	2816‖	900/–/–	16/1024/–	235	Q3 2014
HD 7970	AMD	GPU	Tahiti	2048‖	925/1010/–	16/768/–	250	Q4 2011
R9 290X	AMD	GPU	Hawaii	2816‖	1000/–/–	16/1024/–	250	Q3 2014
R9 295x2	AMD	GPU	Hawaii	5632‖	1018/–/–	16/1024/–	500	Q2 2014
R9 Fury X	AMD	GPU	Fuji	4096‖	1050/–/–	16/2048/–	273	Q2 2015
RX 480	AMD	GPU	Polaris	4096‖	1120/1266/–	16/2048/–	150	Q2 2016
Xeon Phi 7210	Intel	MIC	KNL	256‡	1300/1500/–	32/1024/–	215	Q2 2016

EOD Evaluation

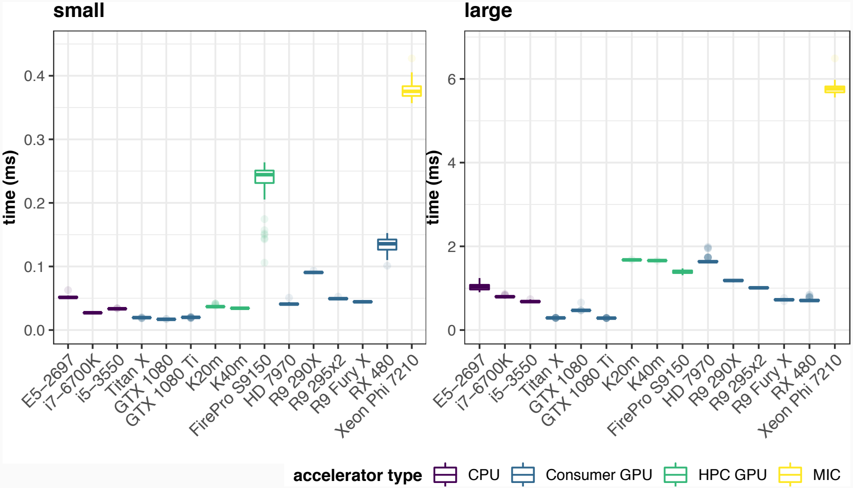


Figure 3: Comparison of performance on 2 sizes of csr application.

EOD Evaluation – Continued

- Just a sample of 1 of 11 applications
- Similar breakdown of 37 kernels
- 15 devices
- Time, performance events and energy (x50)
- Many more results and discussions presented in the paper

What now?

- Small benchmark suite
- Wide diversity of scientific application codes
- Forms a large range of result times (on 6 years of hardware)

What now?

- Small benchmark suite
- Wide diversity of scientific application codes
- Forms a large range of result times (on 6 years of hardware)

Now, forms a test-bed to examine:

1. workload characterization
2. performance prediction
3. scheduling

What now?

- Small benchmark suite
- Wide diversity of scientific application codes
- Forms a large range of result times (on 6 years of hardware)

Now, forms a test-bed to examine:

1. workload characterization
2. performance prediction
3. scheduling

First, how do we characterize a workload?

Workload Characterization

- Many heterogeneous accelerators running OpenCL in current and future supercomputers, and this trend is continuing
- **But** their performance is as diverse as each accelerators hardware configuration
- An architecture-independent method to characterize OpenCL codes allows us to:
  - measure inherent program behaviour
  - perform accurate performance predictions for scheduling

Workload characterization with AIWC

- Architecture-Independent Workload Characterization (AIWC)
- Plugin for OclGrind – an Extensible OpenCL device simulator[8]
- Beta available – https://github.com/BeauJoh/Oclgrind – and will be merged into default OclGrind

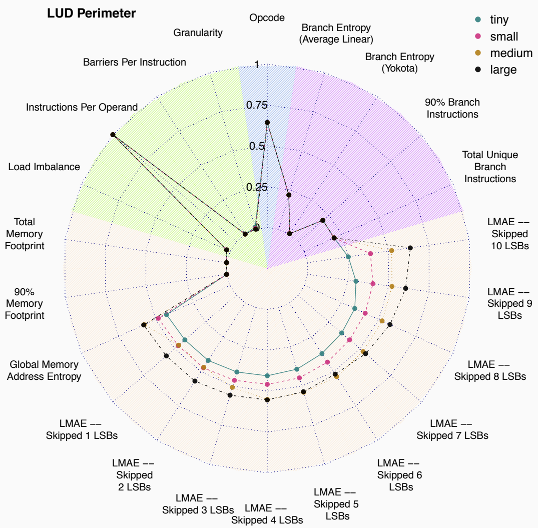
[8] J. Price and S. McIntosh-Smith, "Oclgrind: An extensible opencl device simulator," in Proceedings of the 3rd International Workshop on OpenCL, 2015, p. 12.

Overview of AIWC

- Simulation of OpenCL kernels occur on LLVM IR – SPIR
- AIWC tracks and measures hardware agnostic events
- Metrics carefully selected and collected during simulator execution
- Large number of metrics collected (28)
- Over a wide spectrum computation, thread communication and memory access patterns
- Supports parallel workloads
- Accessible – as part of OclGrind
- High-accuracy – full resolution, not interrupt/sample driven

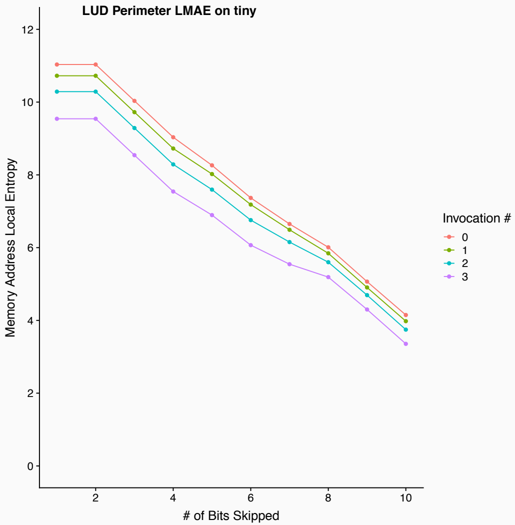
AIWC Example

- Four major classes: Compute, Parallelism, Memory, Control
- Different statistics per metric – distributions, entropy and absolute counts

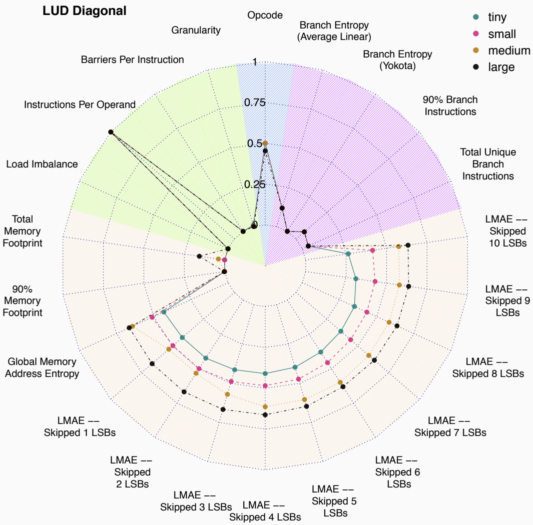


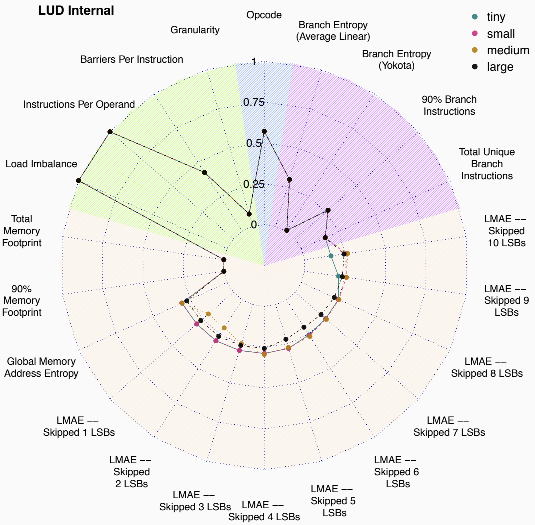
AIWC Example II

- Local Memory Address Entropy
- Kernel launched 4 times – over different problem sizes
- Starting entropy changes with problem size, but same gradient → memory access patterns are the same regardless of actual problem size
- Steeper descent → more localised memory access → better cache utilization



AIWC Example III





Subset of AIWC Metrics

Table 3: Sample of metrics collected by AIWC – ordered by type.

Type	Metric	Description
Compute	Opcode	total # of unique opcodes required to cover 90% of dynamic instructions
Compute	Total Instruction Count	total # of instructions executed
Parallelism	Work-items	total # of work-items or threads executed
Parallelism	Total Barriers Hit	total # of barrier instructions
Parallelism	Median ITB	median # of instructions executed until a barrier
Parallelism	Max IPT	maximum # of instructions executed per thread
Parallelism	Mean SIMD Width	mean # of data items operated on during an instruction
Memory	Total Memory Footprint	total # of unique memory addresses accessed
Memory	90% Memory Footprint	# of unique memory addresses that cover 90% of memory accesses
Memory	Unique Read/Write Ratio	indication of workload being (unique reads / unique writes)
Memory	Reread Ratio	indication of memory reuse for reads (unique reads/total reads)
Memory	Global Memory Address Entropy	measure of the randomness of memory addresses
Memory	Local Memory Address Entropy	measure of the spatial locality of memory addresses
Control	Total Unique Branch Instructions	total # of unique branch instructions
Control	90% Branch Instructions	# of unique branch instructions that cover 90% of branch instructions
Control	Yokota Branch Entropy	branch history entropy using Shannon's information entropy
Control	Average Linear Branch Entropy	branch history entropy score using the average linear branch entropy

AIWC Usage

```
oclgrind --aiwc ./kmeans -p 0 -d 0 -t 0 -- -g -p 256 -f 30
```

- The collected metrics are logged as text in the command line interface during execution and also in a csv file, stored separately for each kernel and invocation.
- Files can be found in the working directory with the naming convention `aiwc_`α`_`β`.csv`.
- Where α is the kernel name and β is the invocation count – the number of times the kernel has been executed.

AIWC Overheads

Table 4: Overhead of the **AIWC** tool on the `fft` benchmark and the Intel i7-6700K CPU.

	time			memory		
	usage (ms)		increase	usage (MB)		increase
	without AIWC	with AIWC		without AIWC	with AIWC	
tiny	0.04	73.4	≈1830×	80.0	85.9	1.07×
small	0.2	427.8	≈2800×	75.9	149.0	1.96×
medium	2.9	12420	≈4300×	101.4	636.8	6.28×
large	19.6	69300	≈3540×	203.8	2213.2	10.86×

- large problems, may exmaine fewer iterations, or run on machines with more virtual memory.
- Envisaged use is that AIWC is only run once, for instance, to examine the characteristics of the kernel in order to identify suitability for accelerators or verify that a high degree of SIMD vectorization had been achieved.

AIWC Example IV

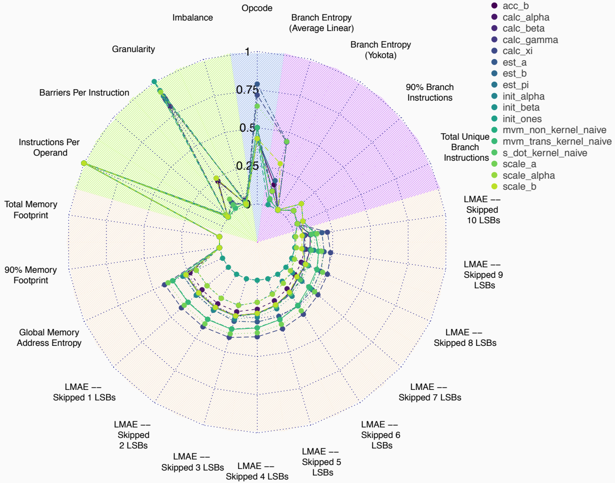


Figure 8: Highlighting the variance in features between kernels in the hmm benchmark.

A larger AIWC Corpus

- Just a sample of 2 of 11 applications
- Similar breakdown of 37 more kernels from the remainder of EOD suite
- Presented in a docker & jupyter artefact

Sample of the AIWC Corpus

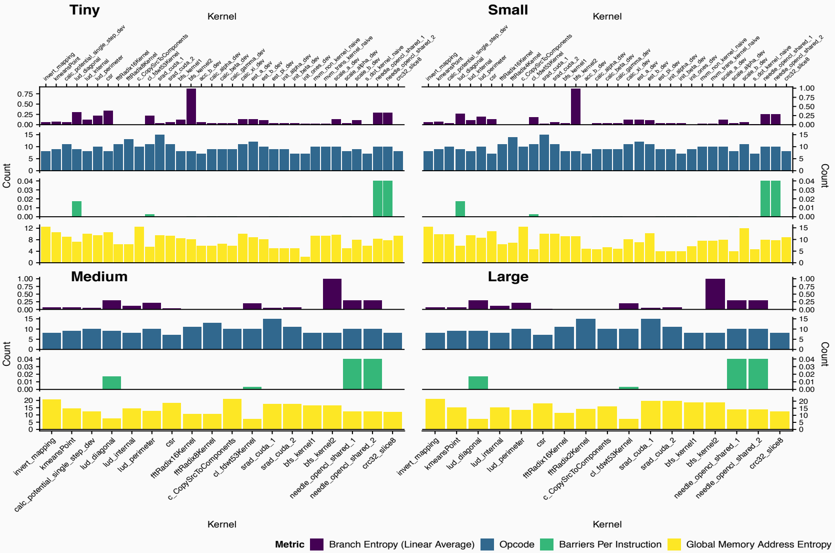


Figure 9: Selected AIWC metrics from each category over all kernels and 4 problem sizes.

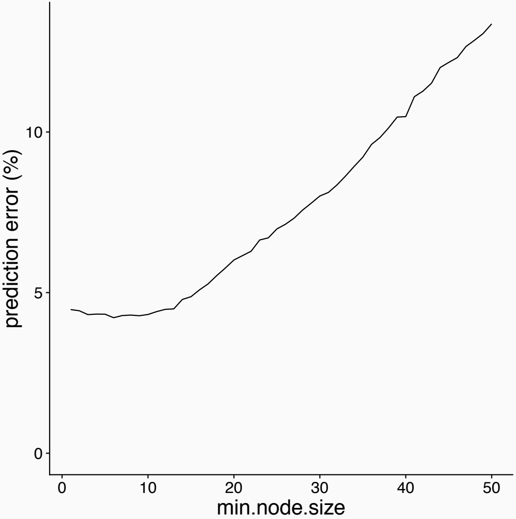
What now?

- AIWC tracks and measures hardware agnostic events
- Large number of metrics collected (28)
- Over a wide spectrum computation, thread communication and memory access patterns
- EOD provides large set of execution times
- AIWC on EOD gives workload characteristics of the same benchmarks
- Forms a large range of result times with matched AIWC feature-spaces
- Can they be coupled together for a predictive model?

Performance Prediction – Combining EOD and AIWC

- Regression model from AIWC was developed
- Predictor variables: 28 AIWC metrics
- Response variable: time or energy of kernel execution
- R language and Ranger – a Random Forest implementation – was used
- Performs recursive partitioning of high dimensional data
- Accepts 3 parameters:
    - num.trees – number of trees grown in forest – 10-10k by 500
    - mtry – number of features tried to split in a node – 1-34
    - min.node.size – minimal size per tree – 1-50
- Optimal model needs careful tuning

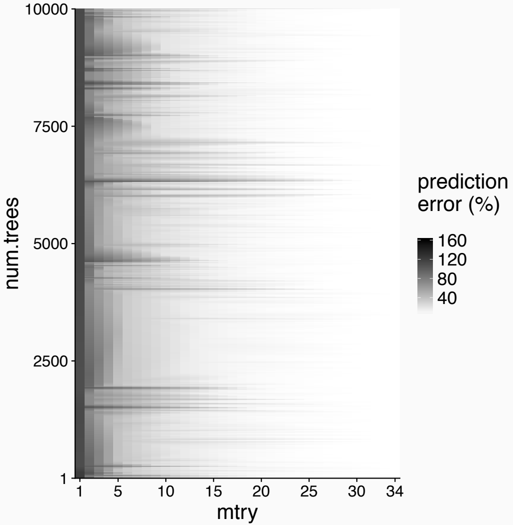
min.node.size



- Full coverage of min.node.size with fixed tuning parameters: num.trees = 300 and mtry = 30
- Smallest out-of-bag prediction error for values < 15
- Selection made to fix min.node.size = 9

num.trees and mtry

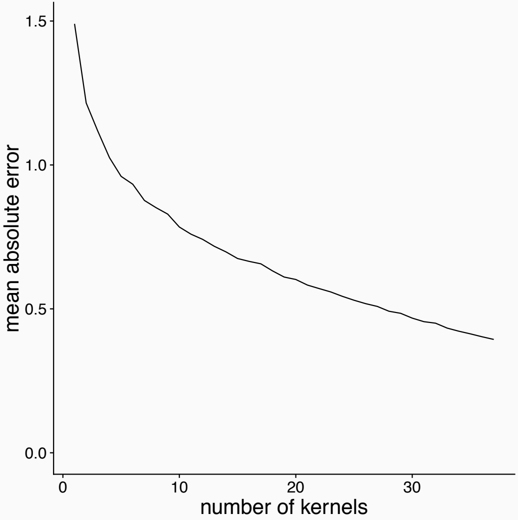
- optim_sa function used to find global minimum
- Full coverage achieved – 4 outer-most points and 8 random starting internal points
- intermediate results used and interpolation performed – using akima
- Model performance varies significantly for last 2 variables
- mtry > 25, offers good fit
- num.trees less impact – fewer trees are computed faster



Choosing Parameters for the Future

- num.trees=500, mtry=32, and min.node.size=9 look good
- train on a random selection of N kernels and test on remainder
- see paper for details but final values are num.trees = 505, mtry = 30 and min.node.size = 9

Increased Training Data



- Prediction error across all benchmarks for models trained with varying numbers of kernels.
- How many kernels to add for training – what's enough?
- Another study performed to see how error changes w.r.t. number of kernels in training
- Uses random selection for each random count – again see paper for full details
- Error tapers off for more kernels!
- gradient still significant at 37 kernels → could still benefit from more.

Prediction Accuracy

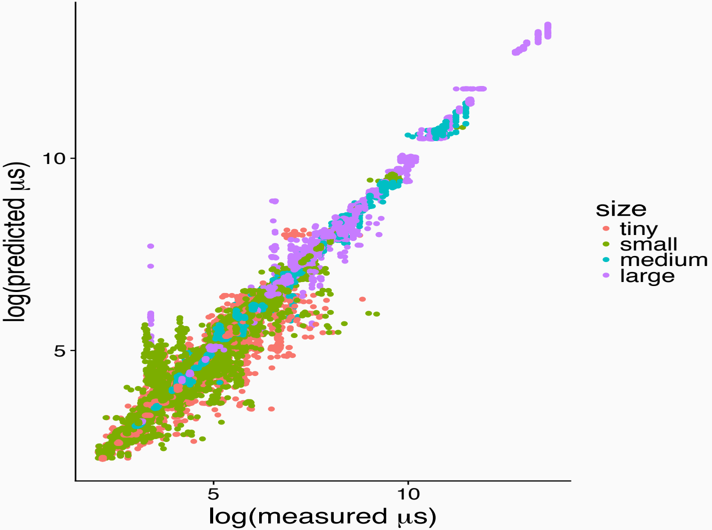


Figure 13: Predicted vs. measured execution time for all kernels.

Prediction Accuracy – Continued

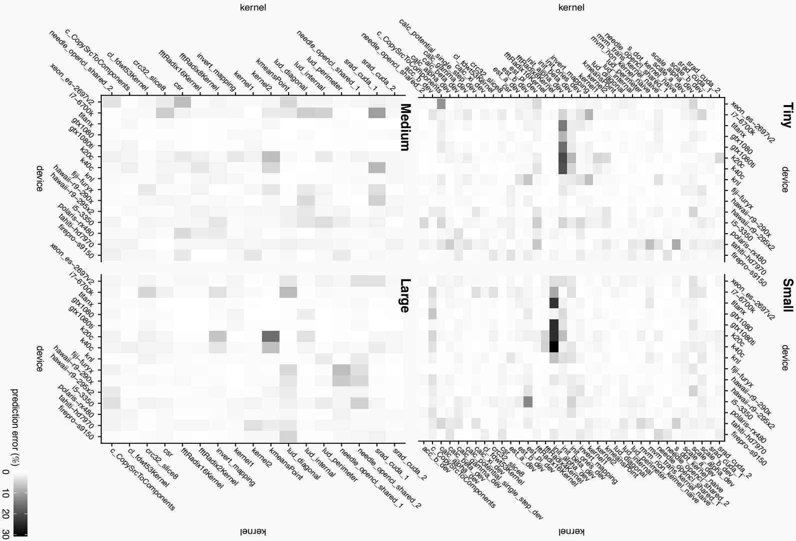
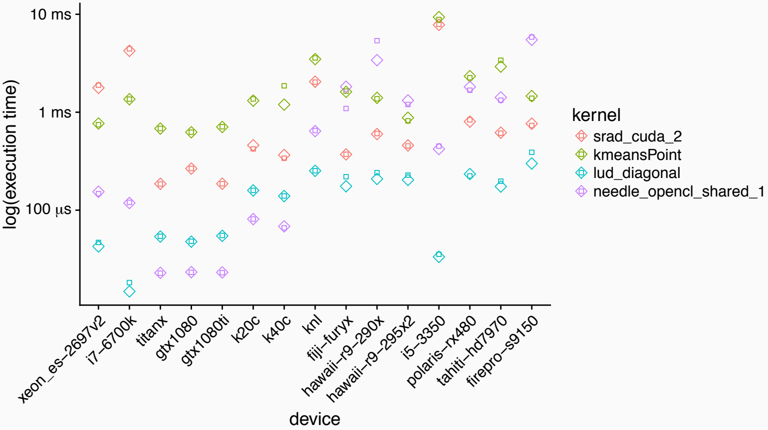


Figure 14: Error in predicted execution time for each kernel invocation over all problem sizes.

Predicting for Scheduling



- Mean measured vs predicted kernel execution
- 4 selected kernels on large problem size
- square → mean measured time
- diamond → mean predicted time
- order is important!

Prediction & Schedulers

- Proposed workflow:
    1. Developer uses Oclgrind to debug, optimize and confirm functionality of a kernel,
    2. Rerun Oclgrind with AIWC to acquire the metrics for the final kernel (with the program settings) that will be used at runtime,
    3. Metrics are included as a comment into the kernel – either in source or SPIR form.
    4. Scheduler extracts metrics at runtime and evaluates them with the model to make performance predictions on the nodes available devices.

Prediction & Schedulers – Considerations

- Modern schedulers (StarPU, Ompss and CoreTSAR) run a new kernel on all devices → Wasteful!
- Our methodology only requires measurements to be taken once – with AIWC – at the time of development
- Model only needs to be retrained when the HPC system is updated, e.g. a new accelerator device, updated driver or compiler.
- Model training is also largely automatic following our prediction methodology, and can use an online corpus – potentially updated by the community of accelerator manufacturers.

Prediction & Schedulers – Considerations

- Modern schedulers (StarPU, Ompss and CoreTSAR) run a new kernel on all devices → Wasteful!
- Our methodology only requires measurements to be taken once – with AIWC – at the time of development
- Model only needs to be retrained when the HPC system is updated, e.g. a new accelerator device, updated driver or compiler.
- Model training is also largely automatic following our prediction methodology, and can use an online corpus – potentially updated by the community of accelerator manufacturers.

But wait! What about: Autotuning and Different problem sizes?

Prediction & Schedulers – Considerations

- Modern schedulers (StarPU, Ompss and CoreTSAR) run a new kernel on all devices → Wasteful!
- Our methodology only requires measurements to be taken once – with AIWC – at the time of development
- Model only needs to be retrained when the HPC system is updated, e.g. a new accelerator device, updated driver or compiler.
- Model training is also largely automatic following our prediction methodology, and can use an online corpus – potentially updated by the community of accelerator manufacturers.

But wait! What about: Autotuning and Different problem sizes?

Schedulers have to deal with incorrectly configured kernels and we **only** target the low-hanging-fruit. If predictions deviate significantly from the measured performance, can fall back to the old run-anywhere approach for given kernel.

Conclusions – EOD

- Completed essential curation of the OpenDwarfs benchmark suite:
  - Increased application diversity
  - Tested on 15 devices
  - High precision measurements of time, energy and hardware events
  - Laid ground work for autotuning

Future Work – EOD

- Autotuner integration – others [9] [10] [11] have shown good performance.
- What does that mean for the scheduling workflow?
- Use the predictive model for synthetic benchmark study

[9] N. Chaimov, B. Norris, and A. Malony, "Toward multi-target autotuning for accelerators," ICPADS, 2014, pp. 534–541.

[10] C. Nugteren and V. Codreanu, "CLTune: A generic auto-tuner for OpenCL kernels," MCSoC, 2015, pp. 195–202.

[11] J. Price and S. McIntosh-Smith, "Analyzing and improving performance portability of opencl applications via auto-tuning," IWOCL, 2017, p. 14.

Conclusions – AIWC

- Presented Architecture-Independent Workload Characterization tool:
  - Supports the collection of architecture-independent features of OpenCL application kernels.
  - First workload characterization tool to support multi-threaded and/or parallel workloads.
- Features can be used to:
  - predict the most suitable device for a particular kernel – for scheduling,
  - or to determine the limiting factors for performance,
  - allows developers to try alternative implementations (e.g. by reorganizing branches, eliminating intermediate variables . . .).
  - inform accelerator designers & integrators of a scientific workloads – ensures compute architectures are suitable for intended workloads.
- AIWC metrics generate a comprehensive feature-space representation → permits cluster analysis and comparison with the dwarf taxonomy.

Future Work – AIWC

- Large working memory fix – write traces to disk instead of linked-list on RAM
- Prune features – redundancy good for random-forests but bad for developers
- Guide a developer to assess:
    - how algorithmic changes → broader characteristics – e.g. no device benefits from more sporadic memory accesses
    - performance portability
    - predicting execution time without having access to these systems (or time to test)
- Language-Agnostic and Architecture-Independent Workload Characterization

Conclusions – Prediction

- Presented highly accurate model for predicting execution times of OpenCL kernels.
- The predictions are highly accurate, differing from the measured experimental run-times by an average of only 1.2%
- Correspond to execution time mispredictions of 9 μs to 1 sec according to problem size.
- Previously unseen code can be instrumented once, AIWC metrics embedded for quick performance prediction.
- Same methodology could be applied to energy usage and hardware events.

Future Work – Scheduling

- Scheduler integration – StarPU, Ompss, CoreTsar or AutoMatch?
- Potential to be more prescriptive than the Berkeley Dwarf Taxonomy

Thesis Statement Revisited

“Scientific high performance computer systems are becoming increasingly heterogeneous because certain applications are more suitable to particular accelerators. The characteristics of a program are sufficient to predict its performance which will allow the efficient scheduling of work to the most appropriate accelerator.”

Open Questions and Next Steps

- Do you have unusual/real-world applications:
  - that you think we've missed?
  - large applications that could benefit from predictions?
  - where you think the predictions will fail?
- Quickly test with Docker & Jupyter artefact.
- Building a prototype scheduler using our predictions?
- Access/Interest in other hardware?
- What does this mean for FPGAs?
- Interested in OpenMP, OpenACC or SYCL evaluation or extensions for AIWC?

Open Questions and Next Steps – Continued.

- Have suggestions for more/better AIWC metrics – or think we've missed some?
- Interested in presenting a reduced feature space?
- Think this descriptive tool to guide developers would be useful? Your input would be appreciated!
- Have applications with weird optimization results? Will changes in metrics show this?
- Anyone an expert in OpenACC back-end – translation to OpenCL kernels for Oclgrind?
- We should collaborate!

Publications

- Johnston B and Milthorpe J “AIWC: OpenCL-based Architecture-Independent Workload Characterisation”, LLVM-HPC workshop, SC18, Dallas, Texas, USA, 2018.
- Johnston B and Milthorpe J “Dwarfs on Accelerators: Enhancing OpenCL Benchmarking for Heterogeneous Computing Architectures”, ICPP, Eugene, Oregon, USA, 2018.
- Johnston B, Falzon G and Milthorpe J “OpenCL Performance Prediction using Architecture-Independent Features”, HPCS, Orleans, France, 2018.
- Johnston B and McCreath E “Parallel Huffman Decoding: Presenting Fast and Scalable Algorithm for Increasingly Multicore Devices”, Guangzhou, China, ISPA 2017.
- Johnston B, Lee B and Angove L and Rendell A “Embedded Accelerators for Scientific High-Performance Computing: An Energy Study of OpenCL Gaussian Elimination Workloads”, Bristol, UK, ICPPW 2017.

Thanks

- The University of Bristol's High Performance Computing Research group for the use of "The Zoo" Research cluster
- Family, friends and supervisors
- You!

beau.johnston@anu.edu.au