ZettaBricks: A Language, Compiler, and Runtime Environment for Anyscale Computing

Saman Amarasinghe, Jason Ansel, Alan Edelman, and Una-May O'Reilly
Massachusetts Institute of Technology
http://projects.csail.mit.edu/petabricks/

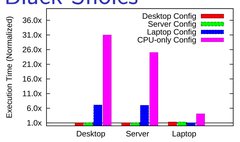
Heterogeneous Choices

Configuration Summaries

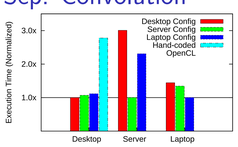
	Desktop Config	Server Config	Laptop Config
Black-Sholes	100% on GPU	100% on OpenCL	Concurrently 25% on CPU and 75% on GPU
Poisson2D SOR	Split on CPU followed by compute on GPU	Split some parts on OpenCL followed by compute on CPU	Split on CPU followed by compute on GPU
SeparableConv.	1D kernel+local memory on GPU	1D kernel on OpenCL	2D kernel+local memory on GPU
Sort	Polyalgorithm: above $\mathbf{174762}$ 2MS (PM), then QS until $\mathbf{64294}$, then 4MS until $\mathbf{341}$, then IS on CPU[1]	Polyalgorithm: above $\mathbf{7622}$ 4MS, then 2MS until $\mathbf{2730}$, then IS on CPU[1]	Polyalgorithm: above $\mathbf{76830}$ 4MS (PM), then 2MS until $\mathbf{8801}$ (above 34266 PM), then MS4 until $\mathbf{226}$, then IS on CPU[1]
Strassen	Data parallel on GPU	8-way parallel recursive decomposition on CPU, call LAPACK when $< \mathbf{682} \times \mathbf{682}$	Directly call LAPACK on CPU
SVD	First phase: task parallism between CPU/GPU; matrix multiply: 8-way parallel recursive decomposition on CPU, call LAPACK when $< 42 \times 42$	First phase: all on CPU; matrix multiply: 8-way parallel recursive decomposition on CPU, call LAPACK when $< \mathbf{170} \times \mathbf{170}$	First phase: all on CPU; matrix multiply: 4-way parallel recursive decomposition on CPU, call LAPACK when $< 85 \times 85$
Tridiagonal Solver	Cyclic reduction on GPU	Direct solve on CPU	Direct solve on CPU

Summary of the different autotuned configurations for each benchmark, focusing on the primary differences between the configurations. [1]For sort we use the abbreviations: IS = insertion sort, 2MS = 2-way mergesort, 4MS = 4-way mergesort, QS = quicksort, PM = with parallel merge.

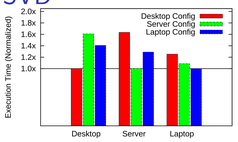
Black-Sholes



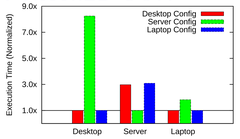
Sep. Convolution



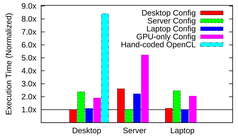
SVD



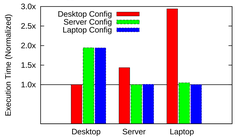
Poisson2D SOR



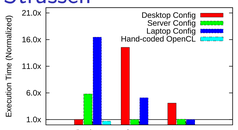
Sort



Tridiagonal Solver



Strassen



Benchmark performance when varying the machine and the program configuration. Execution time on each machine is normalized to the natively autotuned configuration. Lower is better. Convolution, Sort, and Strassen include *Hand-coded OpenCL* as a baseline taken from the NVIDIA SDK sample code. This baseline uses NVIDIA-specific constructs and only runs on our Desktop system. These hand-coded OpenCL baselines implement 1D separable convolution, radix sort, and matrix multiply respectively. As additional baselines, SOR includes a *CPU-only Config* which uses a configuration autotuned with OpenCL choices disabled and Sort includes *GPU-only Config* which uses PetaBricks bitonic sort on the GPU.

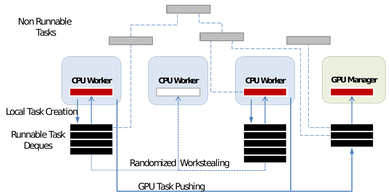
Abstract

ZettaBricks is a language and compiler based on automatic poly-algorithm composition and autotuning which addresses the portability, design complexity, and performance tuning challenges of exascale machines. ZettaBricks automatically builds programs that are able to adapt to their environment. The ZettaBricks language allows the programmer to code different algorithms for the same purpose then, rather than requiring the programmer choose which one is more efficient for each given context, offload the responsibility of finding the best algorithmic choices to the compiler and runtime system. The compiler then generates further choices in how the programmer's different algorithms can map to CPU and GPU processors and memory systems. These choices are given to an empirical autotuning framework that allows the space of possible implementations to be searched at installation time. Â The rich choice space allows the autotuner to construct poly-algorithms that combine many different algorithmic techniques, using both the CPU and the GPU, to obtain better performance than any one technique alone.

We have shown how empirical autotuning can automatically determines the best mapping of programs in a high level language across a heterogeneous mix of parallel processing units, including placement of computation, choice of algorithm, and optimization for specialized memory hierarchies. With this, a high-level, architecture independent language can obtain comparable performance to architecture specific, hand-coded programs.

In evaluating our system across a range of different types of single node system we have shown that algorithmic choices are required to get optimal performance across machines with different processor mixes, and the optimal algorithm is very different in different architectures. We refute the conventional wisdom that if using the GPU is viable it is always a win, and instead show that the choice of how and when to use the GPU is much more complex, with no simple answer. Even when using the GPU is a win, the best mapping of a program to the GPU differs between different types of GPUs. Finally, we demonstrate that splitting work to run concurrently both on the GPU and the CPU can significantly outperform using either processor alone. Experimental results show that algorithmic changes, and the varied use of both CPUs and GPUs, are necessary to obtain up to a 16.5x speedup over using a single program configuration for all architectures.

Task Model



Online Evolutionary Autotuner

Test Systems (left)

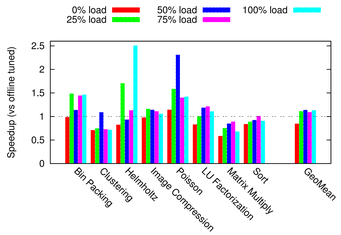
Codename	CPU(s)	Cores	GPU	OpenCL Runtime
Desktop	Core i7 920 @2.67GHz	4	NVIDIA Tesla C2070	CUDA Toolkit 3.2
Server	4× Xeon X7550 @2GHz	32	None	AMD APP SDK 2.5
Laptop	Core i5 2520M @2.5GHz	2	AMD Radeon HD 6630M	Xcode 4.2

Test Systems (right)

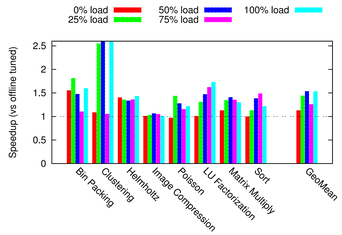
Codename	CPU(s)	Cores
Xeon8	Intel Xeon X5460 @3.16GHz	8
Xeon32	Intel Xeon X7560 @2.27GHz	32
AMD48	AMD Opteron 6168 @1.9GHz	48

Online Auotuning: SiblingRivalry

Adapting to Changing Load

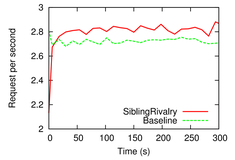


Xeon8

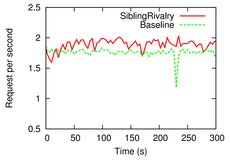


AMD48

Adapting to Architecure Migrations

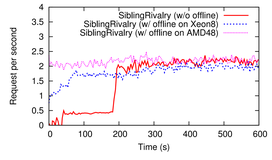


Matrix Multiply, migrate Xeon8 to AMD48



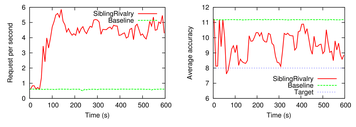
Sort, migrate AMD48 to Xeon32

Starting Configurations

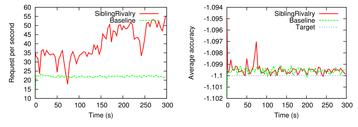


Sort on AMD48

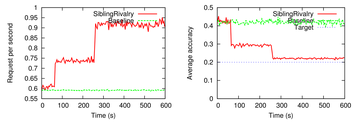
Adapting with Variable Accuracy



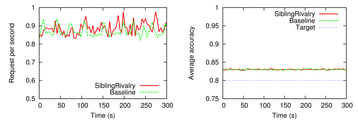
Helmholtz, migrate Xeon8 to AMD48



Bin Packing, migrate AMD48 to Xeon32



Clustering, migrate Xeon8 to AMD48



Matrix Approximation, migrate Xeon8 to AMD48