Modeling and Predicting Application Performance on Hardware Accelerators

Presented by: Alexander Breslow

Authors: Mitesh Meswani*, Laura Carrington*, Didem Unat, Allan Snavely, Scott Baden, and Steve Poole

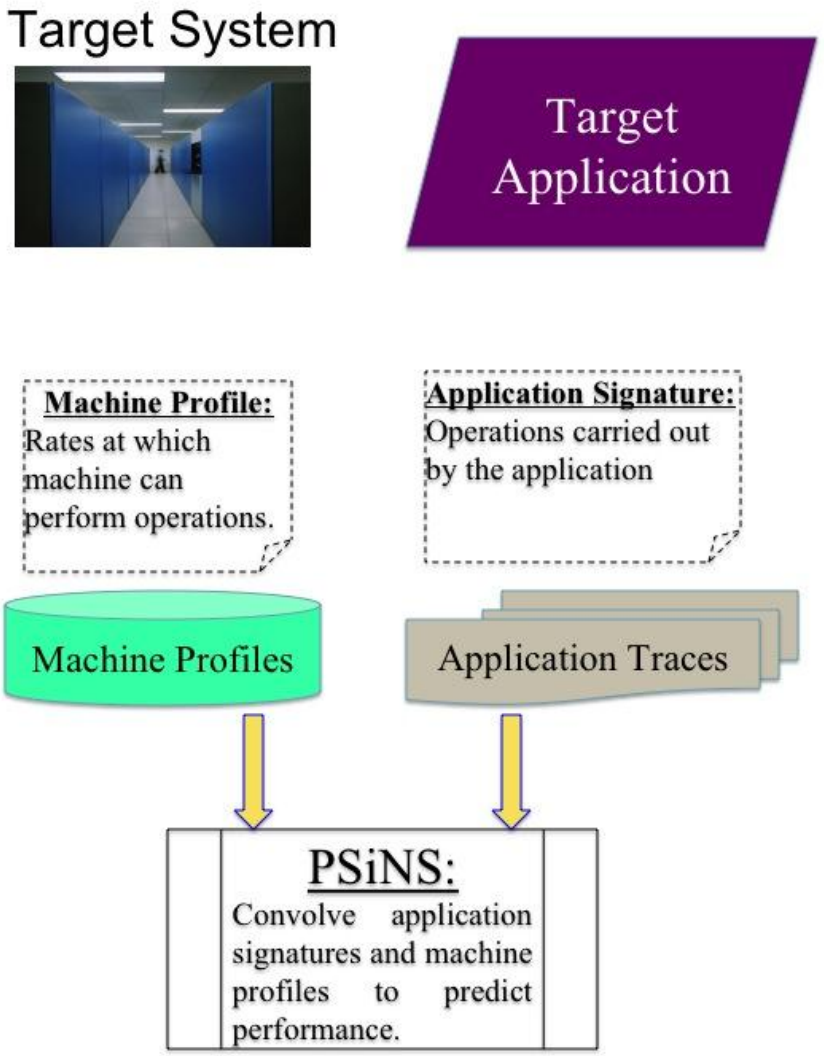
*San Diego Supercomputer Center,

*Performance Modeling and Characterization Lab (PMaC)

PMaC Lab

- **Goal: Understand factors that affect runtime and now recently energy performance of HPC apps on current and future HPC systems**
- **PMaC framework provides fast and accurate predictions**
  - **Input: software characteristics, input data, hardware parameters**
  - **Output: Prediction model that predicts expected performance**
- **Tools : PEBIL, PMaCInst, PSiNSTracer Etracer, IOTracer, ShmemTracer, PIR**
- **Simulation: PSiNS, PSaPP**

Prediction framework



Outline

- **Introduction**
- **Methodology- developing models for FPGAs and GPUs**
- **Results- workload predictions on accelerators**
- **References**

Why Accelerators?

- **Traditional processing**
  - **Solves the common case**
  - **Limited performance for specialized functions**

- **Solution : Use special purpose co-processors or Hardware Accelerators**
  - **Examples: FPGA, GPU**

Application porting is time consuming

- **HPC apps can case exceed 100,000 lines of code**
- **Choice of accelerator is not apparent**
- **Prudent to evaluate benefit prior to porting**

- **<u>Solution:</u> performance predictions models**
  - **Allow fast evaluations without porting or running**
  - **Accuracy has to be high to be valuable**

Methodology

- First identify code sections that may benefit from accelerators
- HPC applications can be expressed by a small set of commonly occurring compute and data-access patterns also called as idioms, example transpose, reduction.
- Predict performance of idiom instances on accelerators.
- Port only instances that are predicted to run faster

Our Study

- **Accelerators: Convey HC-1 FPGA system and NVIDIA FERMI GPU (TESLA 2070)**
- **Characterize accelerators for 8 common HPC idioms**
- **Develop and validate idiom models on two real-world benchmarks.**
- **Present a case study of a hypothetical Supercomputer with FPGAs, GPUs for two popular HPC apps predict speedups up to 20%**

What are Idioms

- **Idiom is a pattern of computation and memory access.**

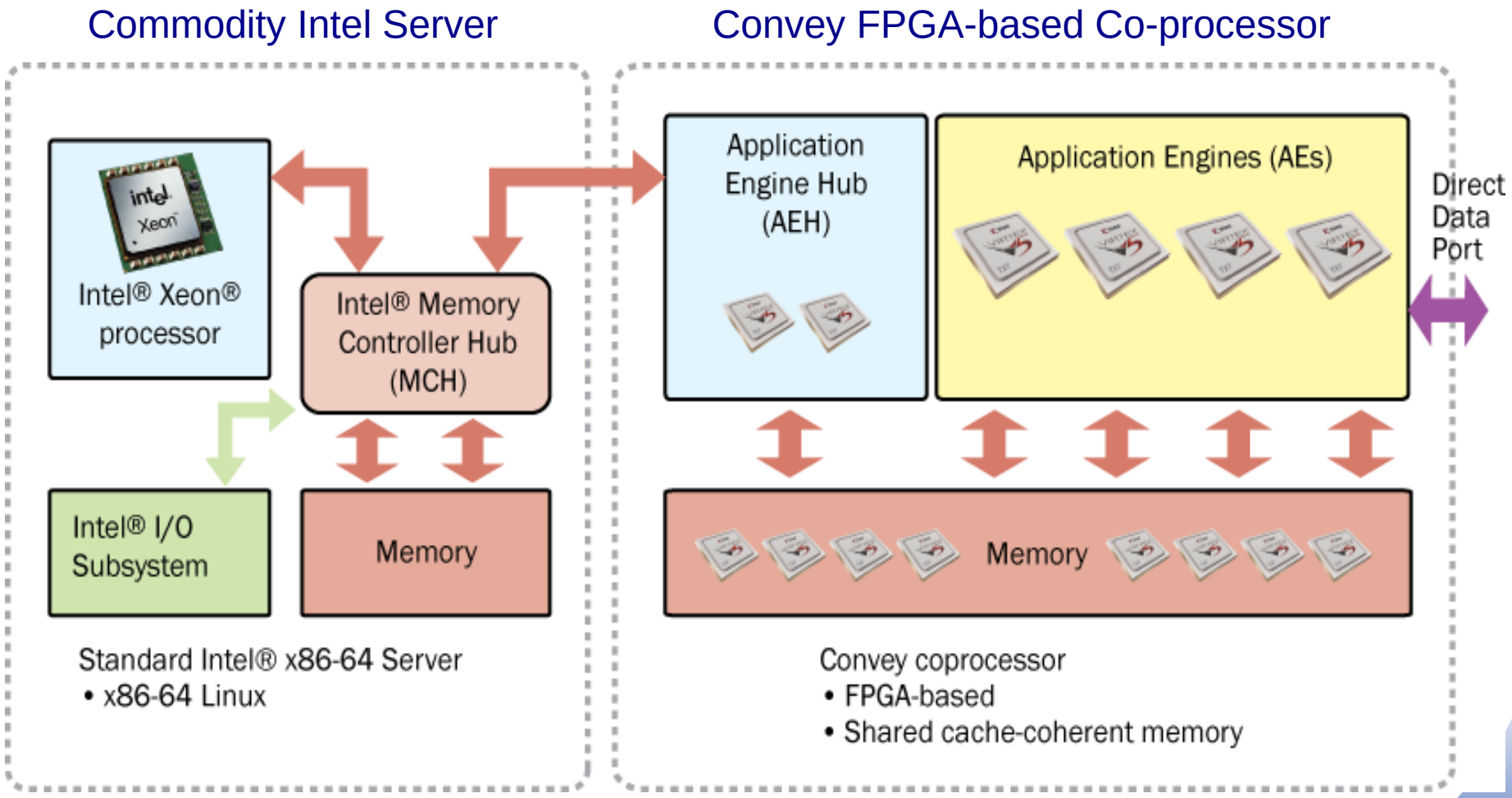
- **Example: Stream copy**

```
for (int i=0;i<n;i++)
  A[i] = B[i] ;
```

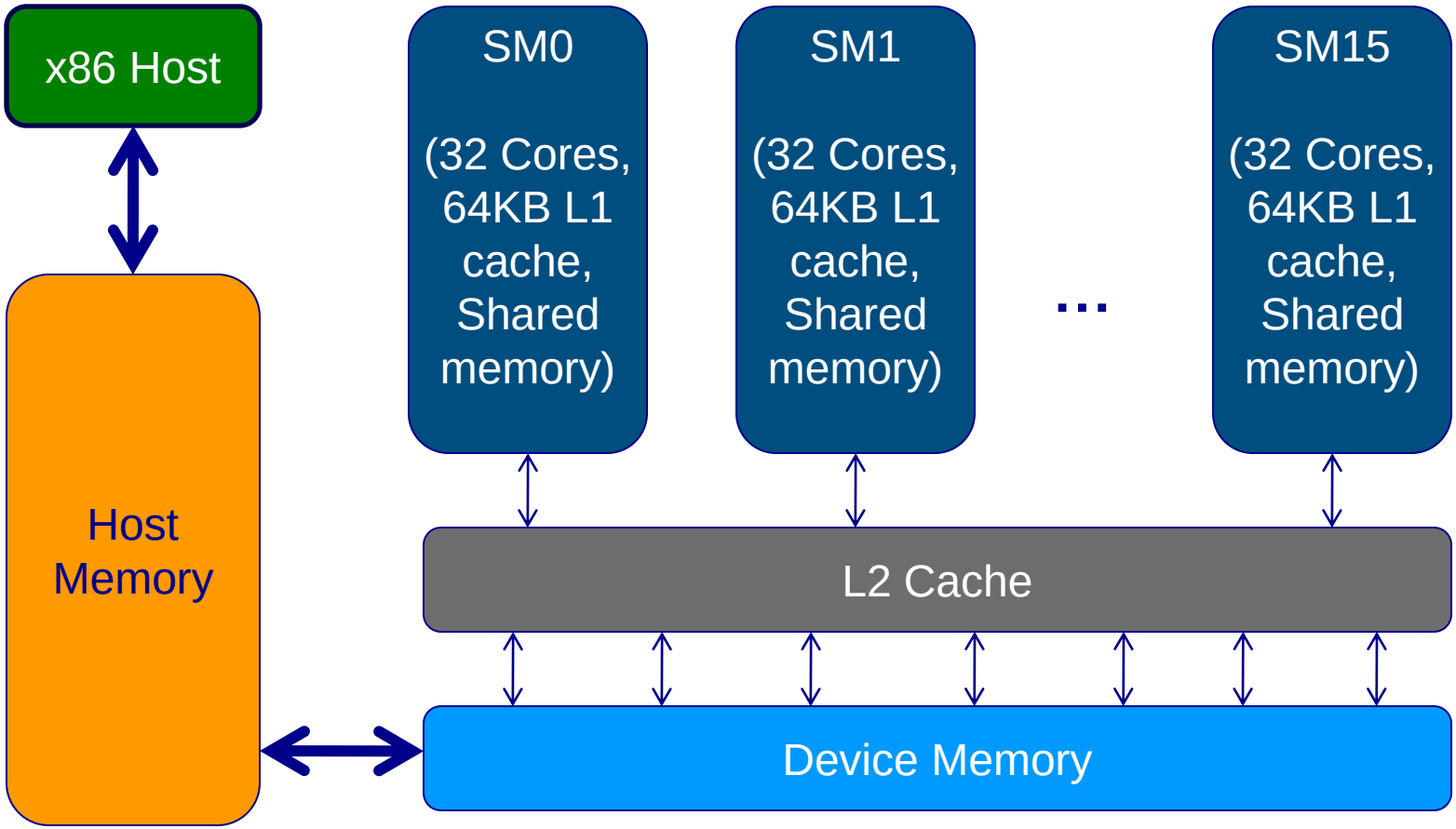
Idioms Used

- Stream : A[i] = B[i] + C[i]
- Gather: A[i] = B[C[i]]
- Scatter: A[C[i]] = B[i]
- Transpose: A[i][j] = B[j][i]
- Reduction: s = s + A[i]
- Stencil: A[i] = A[i-1] + A[i+1]
- Matrix Vector Multiply: C[i] = A[i][j]*B[i]
- Matrix Matrix Multiply: C[i][j] = A[i][j]*B[k][j]

Hardware Accelerator #1 – Convey HC-1 FPGA



Hardware Accelerator #2 – NVIDIA TESLA 2070C GPU

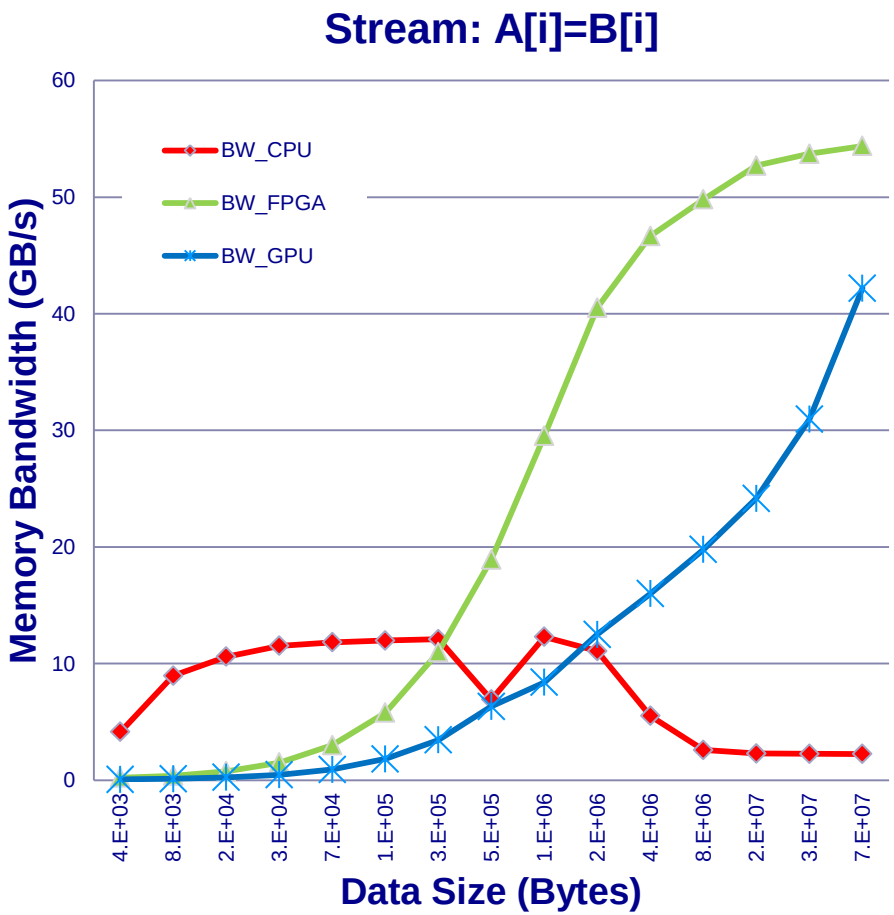


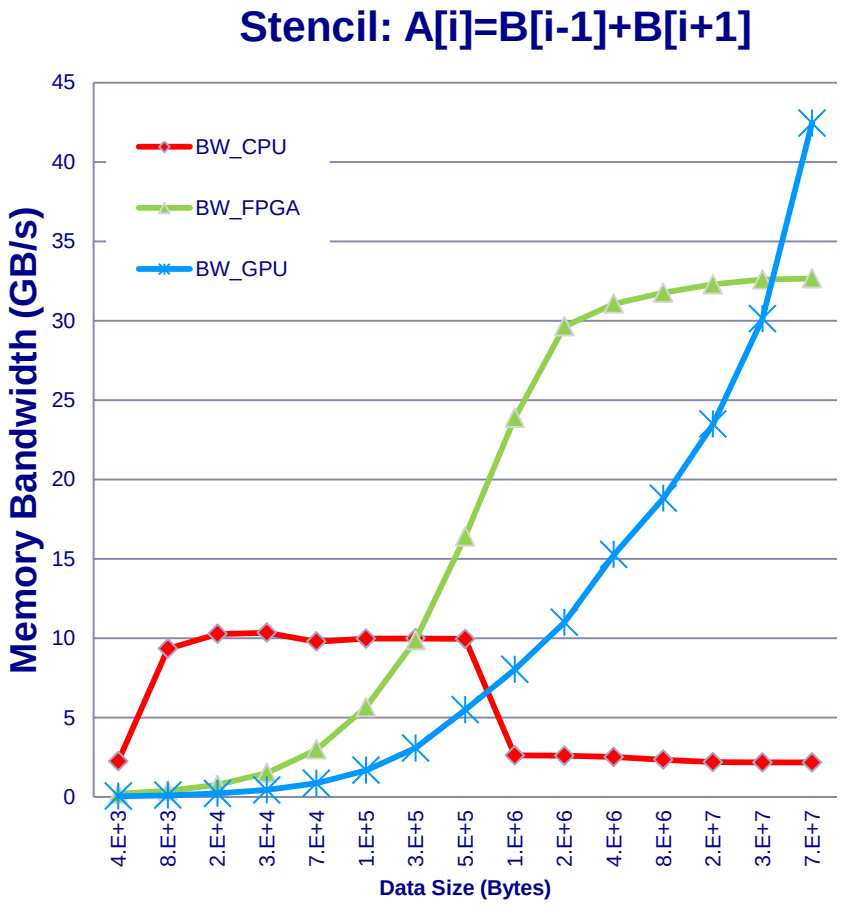
Accelerator Characterizations

- **Simple benchmarks to profile capabilities of GPU, FPGA, and CPU to perform idiom operations**

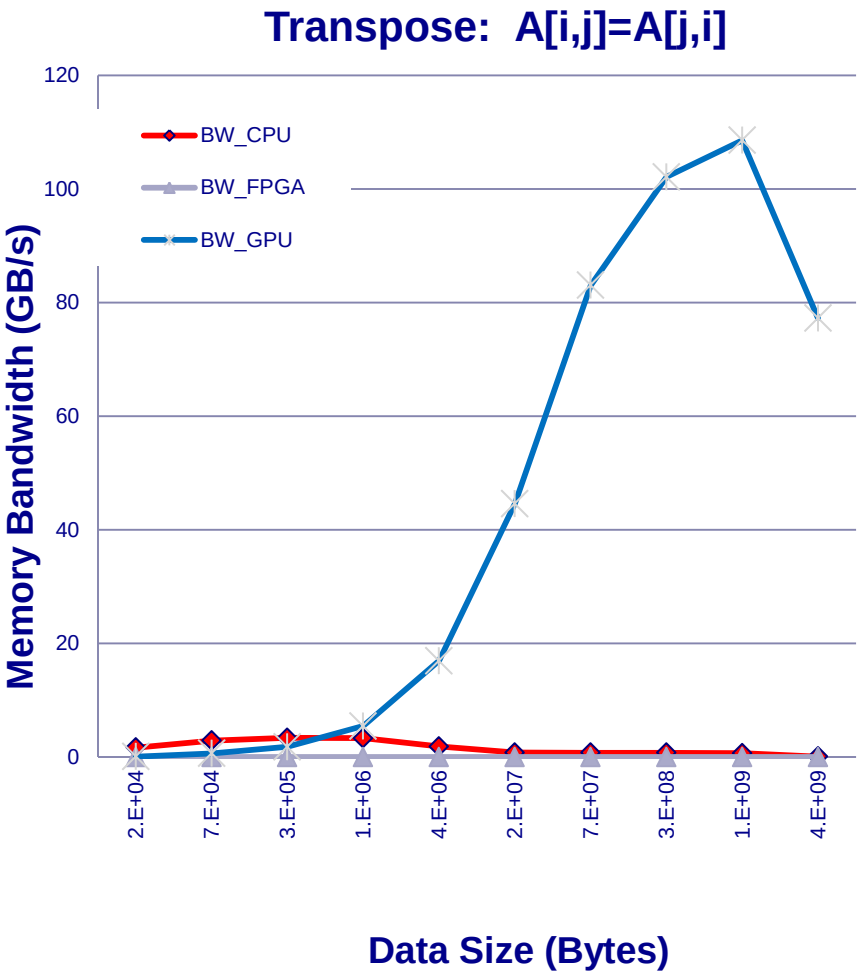
- **Each benchmark ranges over different memory sizes**

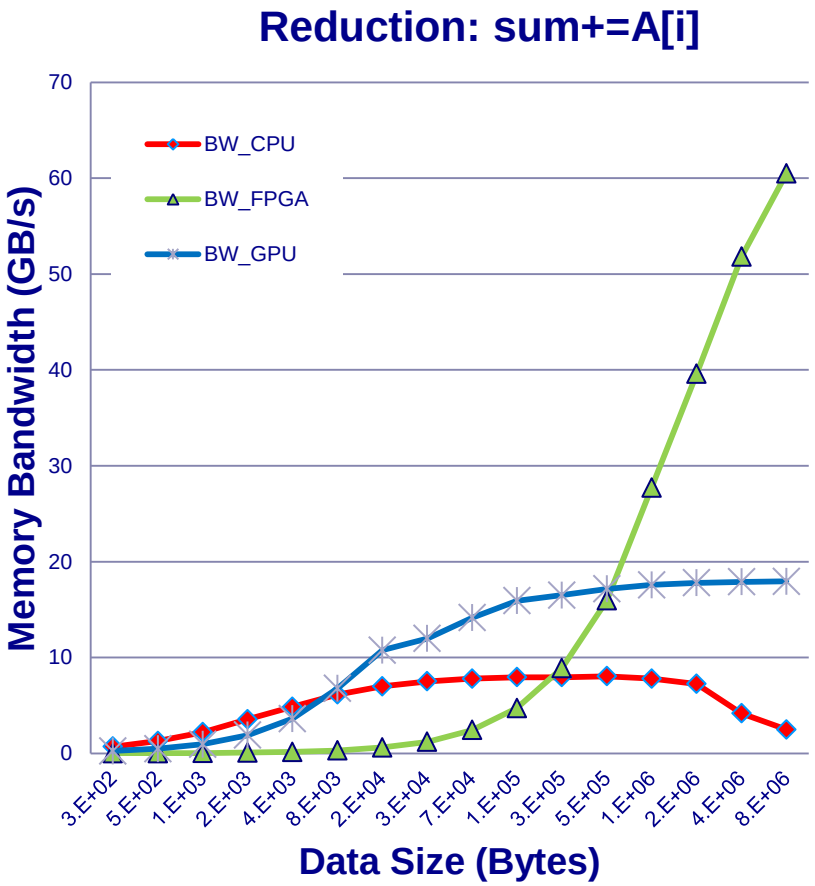
Stream, Stencil





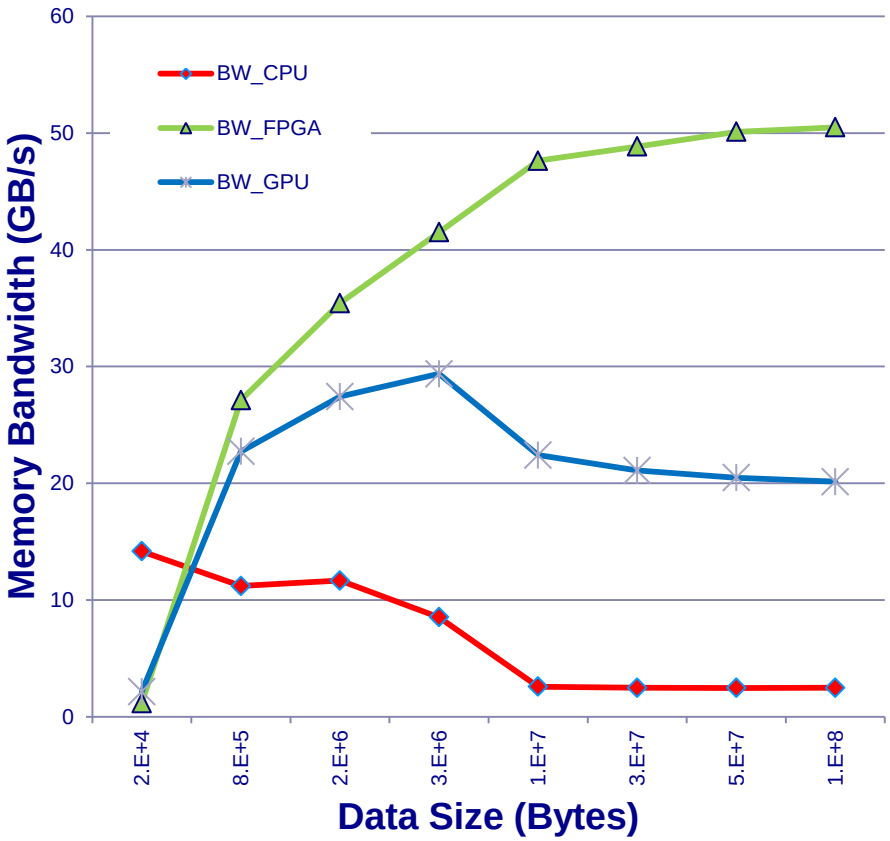
Transpose, Reduction



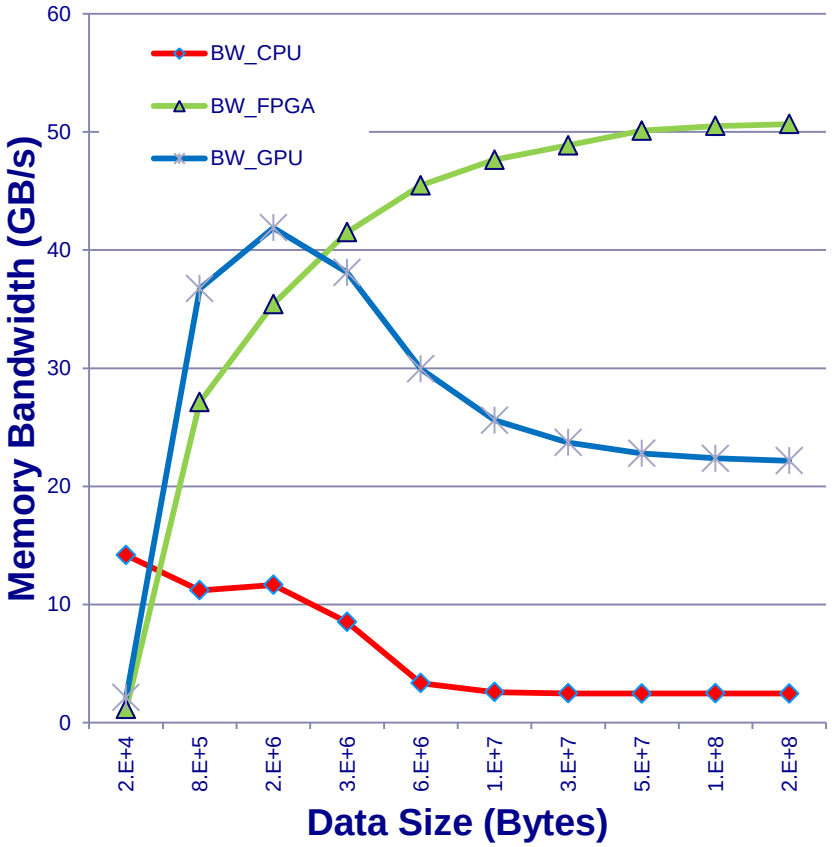


Gather, Scatter

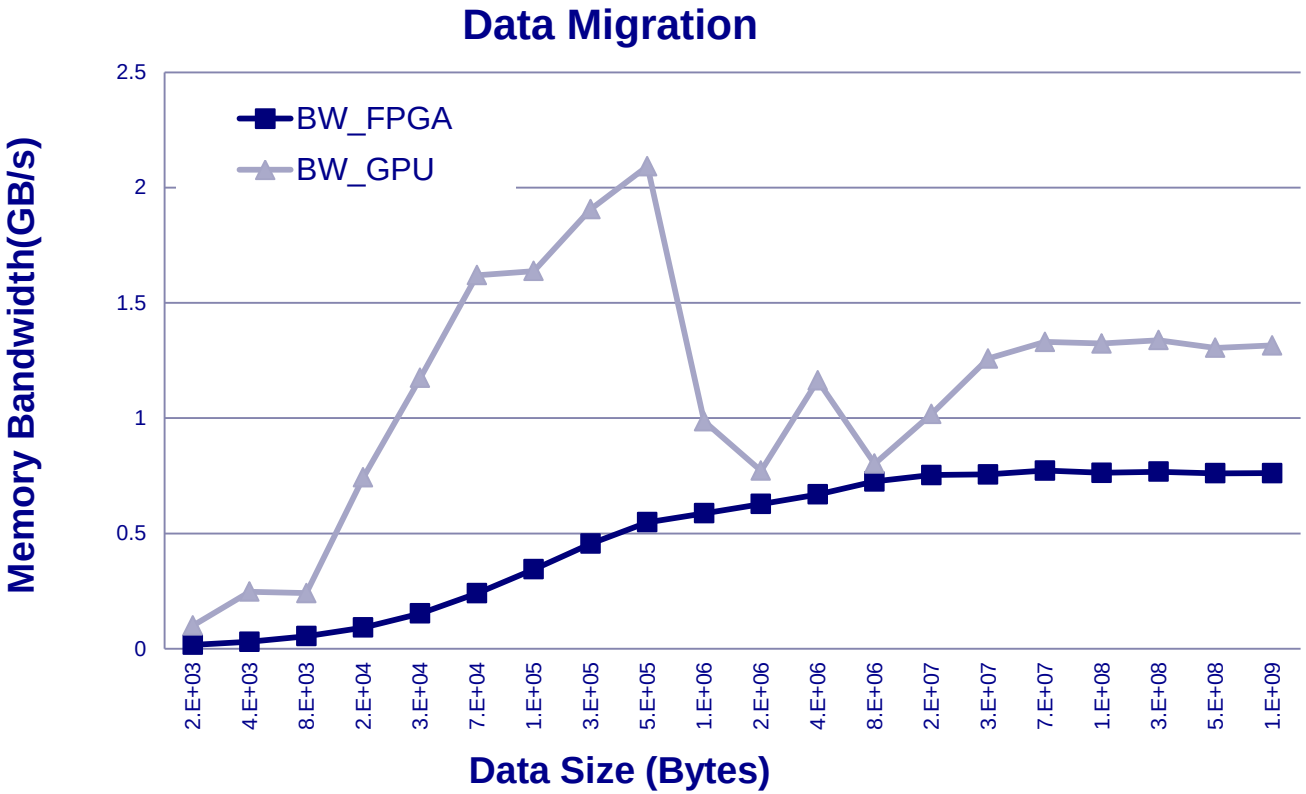
Gather: A[i] = B[C[j]]



Scatter: A[B[i]] = C[j]



Cost of Data Migration



Combining idiom plots and data migration costs illustrates the complexity of determining the best achievable performance from the GPU/FPGA for a given data size and it is interesting to note this space is complex – there is no clear winner among CPU, FPGA, GPU it depends on the idiom and the dataset size.

Application Characterizations – finding idioms

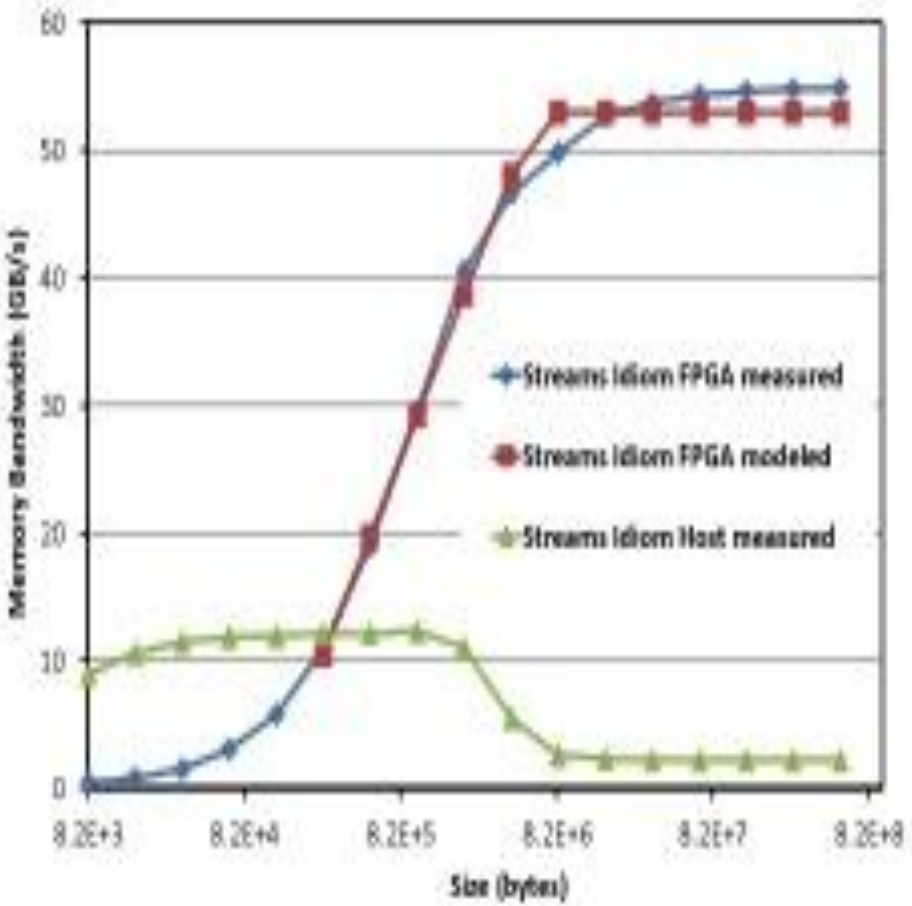
- **PMaC Idiom Recognizer (PIR):**
  - **GCC plugin recognizes idioms during compilation using IR tree analysis**
  - **Users can specify different idioms using PIR's idiom expression syntax**

File	Line#	Function	Idiom	Code
foo.c	623	Func1	gather	a[i] = b[d[j]]
tmp.c	992	Func2	stream	x[j]= c[i]

Application Characterizations – finding data size per idiom

- **PEBIL – binary instrumentation tool**
  - **To find data size for an idiom:**
    - **Determine basic blocks belonging for the idiom**
    - **Instrument those basic blocks to capture data range**
  - **Run the instrumented binary and generate traces**

Prediction Models



Equation 2):

$$Memory\ Time = \sum_{i}^{all\ BB} \frac{\#Mem\ Ref_{i,j} \times Ref\ Size}{Mem\ BW_{stream}}$$

Where,

$Mem\ BW_{stream}$ =

13.59*ln(RefSize) - 159.2, Ref Size <= 4194304,
53.0, RefSize > 4194304

Ref Size = size of the reference in bytes

$Mem\ Ref_{i,j}$ = num of references of basic block i type j

Model Validation – Fine-grained

- **Hmmer: Protein sequence code, run with 8-tasks on GPU, FPGA systems**
- **Flash: astrophysics code, sequential version run on FPGA system.**

Application	Idiom	Measured	Predicted	% Error[1]
Hmmer	Stream (FPGA)	384.7	337.0	12.3%
Hmmer	Stream (GPU)	18.4	18.5	0.3%
Hmmer	Gather/Scatter (GPU)	0.074	0.087	17.3%
Flash	Gather /Scatter (FPGA)	69	68	1.4%

Model Validation – Graph500

- **FPGA validated:**
  - **We ran scale 24 problem, 13 MTEPS**
  - **PIR analysis identifies scatter and stream idiom in make_bfs**
  - **make_bfs ported by convey to FPGA, rest on CPU**
  - **We use CPU and FPGA models to predict speedups**

G500 (CPU) actual	G500 (Ported) actual	Bfs speedup actual	G500 (CPU) predicted	G500 (Ported) predicted	Bfs speedup predicted
5980	4686 (21.64%)	98X	5847	4757 (18.65%)	96x

Projection Study

- **Study production HPC system – Jaguar a Cray XT5**
  - **224,256 AMD cores, 300 TB memory**
- **Applications:**
  - **Hycom – 8 and 256 cpu runs**
  - **Milc – 8 and 256 cpu runs**
- **Q: What would be projected speedup for an appliction running on machine like Jaguar but with FPGA and GPU on each node**

Results – CPU predictions

Application	Measured	Predicted	% Error[1]
Milc (8cpu)	278	277	0.4%
Milc (256cpu)	1,345	1,350	0.4%
HYCOM (8cpu)	262	246	6.1%
HYCOM (256cpu)	809	663	18.1%

Idiom instances and runtime %

Idiom instances in source code

Idiom	HYCOM	Milc
Gather/scatter	1,797	156
stream	1,300	105

Contribution of idioms to runtime

	HYCOM (8cpu)	HYCOM (256cpu)	Milc (8cpu)	Milc (256cpu)
Gather/scatter	14.2%	4.6%	1.2%	0.7%
stream	21.1%	16.9%	5.6%	3.0%

Run times of all idiom instances on one device

HYCOM 256cpu	CPU	FPGA	GPU
Gather/Scatter	7,768	495	638
Stream	28,459	2,302	44,166
Total	**36,556**	**2,798**	**44,803**

MILC 256cpu	CPU	FPGA	GPU
Gather/Scatter	2,376	334	399
Stream	10,452	771	1,087
Total	**12,827**	**1,104**	**1,487**

Optimal mapping of device

HYCOM 256cpu	CPU vs. FPGA	CPU vs. GPU	Optimal of CPU, GPU, FPGA	CPU	FPGA	GPU
Gather/Scatter	495	638	448	7,768	495	638
Stream	2,297	6,096	2,149	28,459	2,302	44,166
Total	2,792	6,734	2,596	**36,556**	**2,798**	**44,803**

MILC 256cpu	CPU vs. FPGA	CPU vs. GPU	Optimal of CPU, GPU, FPGA	CPU	FPGA	GPU
Gather/Scatter	334	399	334	2,376	334	399
Stream	770	1,087	765	10,452	771	1,087
Total	1,104	1,486	1,099	**12,827**	**1,104**	**1,487**

Summary of Results of porting

Overall improvement of porting idioms results in improvements of 3.4% for Milc and 20% for HYCOM.

Conclusions and Future Work

- **Device choice (CPU,GPU,FPGA) depends on idiom and data size**
- **Continue extension to other idioms and model data transfer costs**
- **Adapt the approach for modeling energy consumption**

References

- **M. R. Meswani, L. Carrington, et al., "Modeling and Predicting Performance on Hardware Accelerators," To appear as a *Poster at the International Symposium on Workload Characterization (IISWC)*, 2011.**
- **C. Olschanowsky, M. R. Meswani, et al., "PIR: PMaC's Idiom Recognizer," in *Proceedings of Parallel Software Tools and Tool Infrastructures (PSTI 2010) held in conjunction with ICPP 2010,* San Diego, CA, Sep 2010.**
- **L. Carrington, M. Tikir, et al., "An Idiom-finding Tool for Increasing Productivity of Accelerators," in *Proceedings of Int'l Conf of Supercomputing (ICS),* 2011**

Questions ?

- **Thank you for your attention !**
- **PMaC URL: www.sdsc.edu/pmac**
- **Email: mitesh@sdsc.edu**

Extra Slides

Model Validation – Graph500

- **GPU validation underway:**
  - **Our CPU predictions are within 13%**
  - **CUDA version implemented and measured GPU version speeds up make_bfs 3X-5X**
  - **Optimization continues and is ongoing work**

Model Validation – Entire Application Run

- **Graph500: A data-intensive benchmark**
  - **Two main kernels: make_bfs and verify_bfs**
  - **Briefly the algorithm proceeds in two steps:**
  - **First generates the graph based on the scale factor.**
  - **Second step it samples 64 random key and for each key:**
    - **Executes *make_bfs* to generate parent array, and then**
    - **Executes *validate* routine on the parent array.**
  - **Performance metric called TEPS (travelled edges per second) is calculated using time for *make_bfs***

Idiom Code Coverage

	HYCOM	Milc
Gather/scatter	1797	156
reduction	110	22
stream	1300	105
stencil	132	0
transpose	3986	286
Mat-Mat Mult	2161	6
Mat-Vec Mult	115	2
Fraction of Loops Covered	**67.45%**	**37.44%**

Idiom runtime coverage

	HYCOM (8cpu)	HYCOM (256cpu)	Milc (8cpu)	Milc (256cpu)
Gather/scatter	14.2%	4.6%	1.2%	0.7%
reduction	0.0%	0.1%	15.7%	13.9%
stream	21.1%	16.9%	5.6%	3.0%
stencil	4.7%	11.1%	0.0%	0.0%
transpose	0.9%	2.0%	0.0%	0.0%
Mat-Mat Mult	23.7%	8.6%	61.2%	58.6%
Mat-Vec Mult	0.0%	0.1%	10.5%	16.7%
All Idioms	**64.6%**	**43.4%**	**94.2%**	**93.2%**

Memory Time

(Equation 1):

$$Memory\ Time = \sum_{i}^{all\ BB} \frac{\#Mem\ Ref_{ij} \times Ref\ Size}{Mem\ BW_j}$$

Where,

Mem BW = bandwidth of jth type of reference on target system

Ref Size = size of the reference in bytes

Mem $Ref_{i,j}$ = number of references of basic block i type j