Harmony
An Execution Model and Runtime for Heterogeneous Many Core Systems

Gregory Diamos and Sudhakar Yalamanchili

Georgia Institute of Technology

June 26th, 2008
1. Limitations of Heterogeneous Computing
2. Harmony Architecture
3. Preliminary Results
4. Open Issues
5. Backup
Applications have diverse requirements
Generality - Introduces Inefficiency

CPUs are one size fits all solutions
Specialization

Heterogeneous systems can match algorithms to architectures
But the user must handle scheduling, partitioning, parallelism, and programming.
Runtime support can augment a heterogeneous system
Harmony - Facilitates Core Cooperation

Using dynamically available information
Harmony Contribution

We have seen this before

Kernel scheduling on heterogeneous processors is essentially instruction scheduling on functional units at a higher level

Instruction level parallelism at a higher level

- Heterogeneous Cores - Functional Units
- Compute Kernels - Instructions
- Variables - Registers

We can leverage this prior work
Harmony Programming Model

Variables
- Data objects managed by the runtime

Compute Kernels
- Consume input variables and produces output variables
- Possibly different implementations for different architectures
- Programs execute logically in order, one kernel at a time

Control Decisions
- Consume input variables
- Select the next compute kernel to execute
Harmony Addresses Compatibility With Kernel Libraries

The Problem - Many Configurations
- Systems may have different sets of processors
- Applications should run efficiently on all

A Solution - *Kernel* Libraries
- *Kernel* binaries for at least one common architecture
  - x86 CPU
- Additional binaries can be used if needed

Runtime Mapping
- *Kernel* calls are mapped to binaries prior to execution
Harmony Example

- An application is specified as a sequence of *control decisions* and *compute kernels*
- Kernel calls are nonblocking
- Runtime infers data dependencies
- Speculation across control decisions is possible
Runtime Scheduling

Runtime computes a parallel schedule that minimizes execution time

FA cyclic dependency graph

Parallel schedule on heterogeneous processors
Handling Variable Execution Time

Kernel execution time can be data and system dependent

Impacts the scheduling quality
Handling Variable Execution Time - With Prediction

- Programmer can associate predictor functions with kernels
- Runtime can build regression models over time [1]

```c
31 double matrixMultiplyComplexity( matrix& A, matrix& B )
32{
33
34    return ( double )( A.dim1() * B.dim2() * A.dim2() );
35
36}
```
Initial Implementation

- System Configuration
  - AMD Athlon64 2.4 Ghz, 2GB RAM, Linux 2.6.22, GCC 4.1.3
  - NVIDIA 8800 GT 320MB GPU
- Assumptions
  - No variable renaming
  - No speculative kernel execution

Example Applications

- Matrix multiplication
- Real time ray tracing
- Audio signal analysis
Optimal partitioning for complex applications is often unintuitive. More suited to runtime scheduling than static partitioning by hand.
Matrix Multiplication

Fastest architecture is data dependent

Runtime performance matches that of the fastest architecture for all data sizes
Limitations

Parallelism Model Doesn’t Scale
- Must create a parallel schedule across all processors in the system
- More suitable for scheduling across all types of processors in the system

Still Difficult To Code
- User still must implement kernels for multiple architectures
- Need support from compilers/APIs to target many architectures (CUDA/OpenCL/Tolapai)
Summary

Efficiency vs Complexity
- 17x-100x speedup across all applications tested
- Dynamic behavior and nonstandard programming environments make development intractable

Harmony
- Runtime techniques reduce complexity while maintaining efficiency
- Parallelism is inferred from data dependencies
- Scheduling phase allows adaptation to dynamic behavior
- Maintains compatibility across many system configurations
- Not sufficient alone for massively parallel applications
Acknowledgements

The authors gratefully acknowledge the generous support of this work by LogicBlox Inc. both through research grants as well as technical interactions, and equipment grants from Intel Corp. and NVIDIA Corp.
Examples of Heterogeneous Applications

- Volume rendering on IBM Cell [2]
- Numerical weather prediction on GPUs [3]
- Game servers on GPUs [4]
- N-Body simulations on GPUs [5]
- Thermal conductivity modelling on GPUs [6]
- Euler equations on GPUs [7]
- Ray tracing with KD-tress on GPUs [8]
- AES on FPGAs [9]
Limitations of Heterogeneous Computing
Harmony Architecture
Preliminary Results
Open Issues
Backup

Gregory Diamos and Sudhakar Yalamanchili

Figure 1a. H.264 Decode

Figure 1b. Harmony Program

Figure 1c. Acyclic Dependency Graph

Figure 1d. Parallel Schedule on Heterogeneous Processors
Limitations of Heterogeneous Computing
Harmony Architecture
Preliminary Results
Open Issues
Backup

Gregory Diamos and Sudhakar Yalamanchili
Harmony

![Graph](image)

- **Runtime Overhead (%) vs. Matrix Size**
- **CPU** and **GPU** performance comparison

Matrix Size:
- 16x16
- 32x32
- 64x64
- 128x128
- 256x256
- 512x512
- 1024x1024
- 2048x2048
Limitations of Heterogeneous Computing
Harmony Architecture
Preliminary Results
Open Issues
Backup

Gregory Diamos and Sudhakar Yalamanchili
Harmony
Limitations of Heterogeneous Computing
Harmony Architecture
Preliminary Results
Open Issues
Backup

Gregory Diamos and Sudhakar Yalamanchili
Harmony

![Graph showing runtime overhead (%)](image)
Limitations of Heterogeneous Computing
Harmony Architecture
Preliminary Results
Open Issues
Backup

Gregory Diamos and Sudhakar Yalamanchili

Harmony

Percent Scheduled

Window 1
Frequency Detection
Filter
Window 2
Gen Signal
FFT
Sub band Detection
iFFT
Sample Synthesis

Kernel Name

% CPU
% GPU
Limitations of Heterogeneous Computing
Harmony Architecture
Preliminary Results
Open Issues
Backup

[Diagram showing sequential execution and runtime integration of kernels, with non-blocking procedure calls, dependency resolution, optimization, and synchronization of results.]

