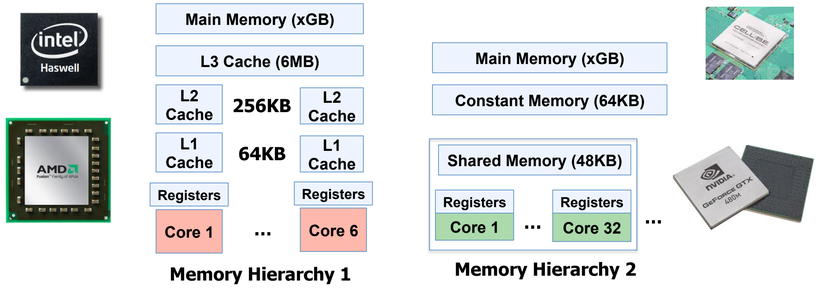
Automatic Data-Layout Framework for Heterogeneous Architectures*

Deepak Majeti (deepak@rice.edu), Kuldeep Meel, Rajkishore Barik and Vivek Sarkar

* Submitted to a conference

Motivation



Heterogeneous architectures have varying memory hierarchies

Implication: Different data layouts for different processors

CPU

Array-Of-Structure (AOS) helps in pre-fetching and cache sharing.

GPU

Structure-Of-Array (SOA) helps in coalescing of memory loads.

"As parallelism goes up, the memory interconnect gets more complex so layout matters, but it is up to the programmer"
--- Norm Rubin from NVIDIA in PPoPP-2014

Current programming models that target heterogeneous architectures require the programmer to specify the data layout.

- Constrains **productivity** and **portability**.

Compilers targeting Heterogeneous Architectures must perform Automatic Data Layout transformation.

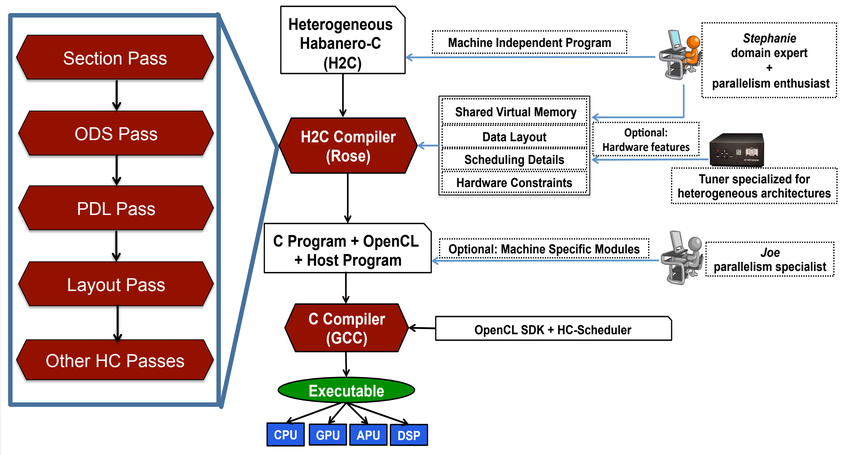
Our Contributions

- Designed an automatic data layout framework for heterogeneous architectures.
- Proved complexity of finding the optimal data layout considering AoS and SoA layouts.
- Implemented the automatic data layout framework on top of Heterogeneous Habanero-C (H2C).
- Showed performance benefits of up to 7x (on average 2.2x) compared to the original layouts.

Framework Design

- Partition a program into sequence of *sections*.
  - A *section* contains kernels which are offloaded to a particular hardware.
- Build affinity graph over the fields accessed in a *section*.
- Optimal Data layout per *Section* (**ODS**) is NP-hard.
  - Heuristic Solution: Clustering heuristic to determine the data layout for a given section.
- How to identify optimal Data Layout for the entire Program (**PDL**) given per section layout ?
  - Optimal Solution: A shortest path problem to identify the data layout optimality for the entire program.
  - A section is mapped only onto one device.

Overall Compilation Framework



- *Section Pass* identifies the *sections* in a given H2C program. A *section* consists of one or more *forasync* constructs.
- *ODS Pass* builds an affinity graph among the fields accessed in a *section* and determines the data layout.
- *PDL Pass* identifies the best layout across all the *sections*.
  - Combines or remaps the layouts of two *sections*.
- *Layout Pass* generates the code corresponding to the data layout inferred.

Experimental Evaluation

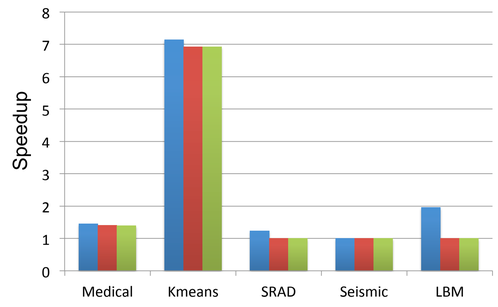
Setup:

Vendor	Type	Model	Freq	Cores	Mem
Intel	CPU	X5660	2.8GHz	12 (HT)	8GB
NVIDIA	Discrete GPU	Tesla M2050	575 MHz	8	2GB

Benchmarks Description:

Name	Description	Original Layout	Num of Kernels	Num of Fields	Input
Registration	Medical Image Registration	SOA	7	6	256 x 256 x 256
SRAD	Speckle Reducing Anisotropic Diffusion	SOA	2	4	4096 x 4096
Seismic	Seismic Wave Simulation	SOA	2	6	10K x 10K
K-Means	Clustering Algorithm	SOA	2	32	8388608
LBM	Computational Fluid Dynamics Simulation	SOA	1	19	300 x 300 x 300

Results:



Speedup **7x** (on average **2.2x**) compared to the original layouts

Conclusions and Future Work

Conclusions:

- Automatic data layout transformation aids programmer productivity and code portability.
- Implemented in H2C and demonstrated upto 7x speedup.

Future Work:

- Analyze more complex data layouts such as AoSoA.
- Sub-partition a section and map onto all available hardware devices.
- Handle control flow between sections.