A Fine-grain Parallel Execution Model for Homogeneous/Heterogeneous Many-core Systems

Jean-Luc Gaudiot

University of California, Irvine

PASCAL: **PA**rallel **S**ystems and **C**omputer **A**rchitecture **L**ab.

University of California, Irvine

PASCAL
University of California, Irvine

Solving the Heterogeneous Many-Core challenge: SPARTA

SPARTA: a Stream-based Processor And Run-Time Architecture

- *Combination of runtime and compiler technologies for a hierarchical heterogeneous many-core chip*
- *Hardware mechanisms for stream-based fine-grain program execution models*
- *Cross-layer methodology (Codelet model combined with generalized streams)*

Based on work performed in cooperation with the University of Delaware (Stéphane Zuckerman and Guang Gao)

The implementation and performance results are from Tongsheng Geng's doctoral dissertation

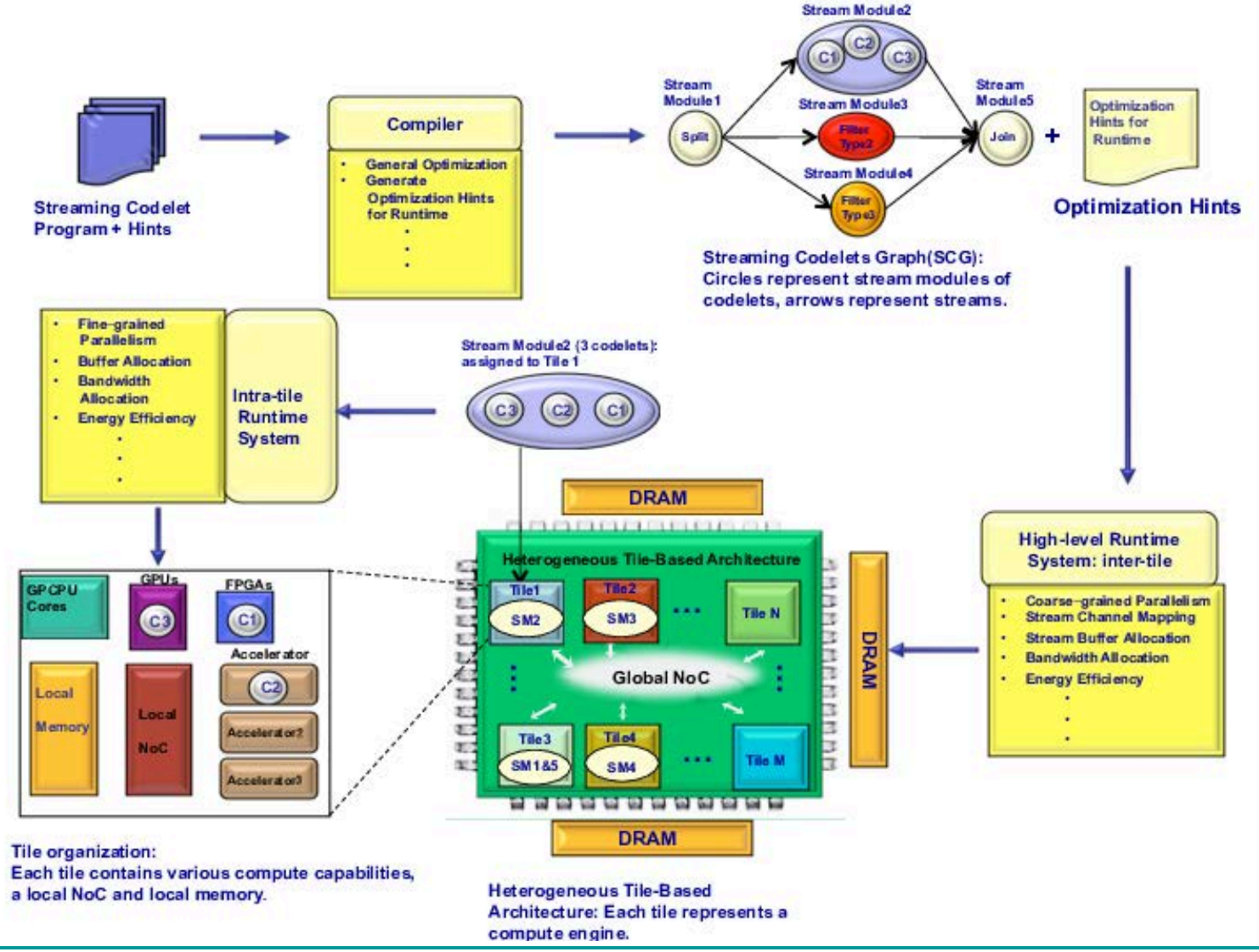
The State of Current High-Performance Computing Systems

- *End of Moore's law and Dennard scaling*
  - *Lasting change in computer architecture: multi and many core systems are here to stay*
- *Current systems feature tens or even hundreds of cores on a single compute node*
  - *Heterogeneous: CPUs, GPUs, FPGAs*
  - *Power and energy aware: homogeneous multi-core substrate may not see cores run at the same clock speed over an application's lifetime, and depending on the workload*
- *Consequence: new programming models (PMs) and execution models (PXMs) must be designed to better exploit this wealth of available parallelism and heterogeneity*

Three main problems to solve

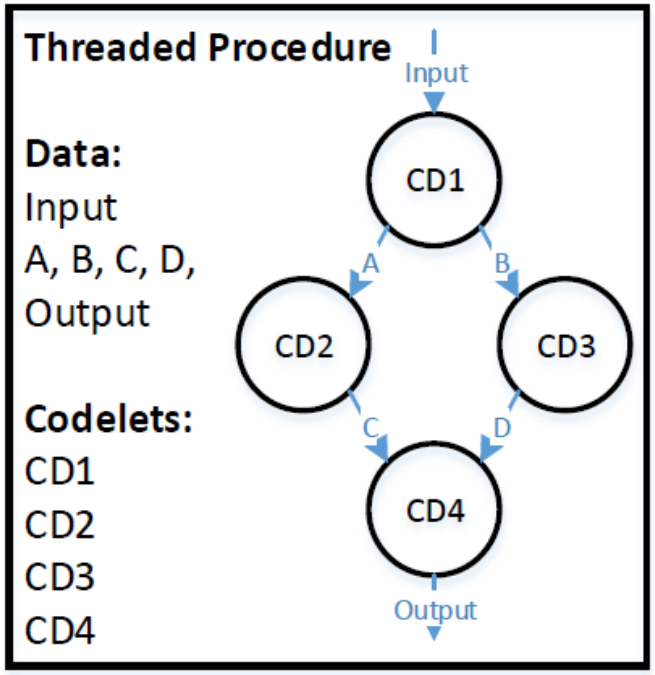
- *Multi-grain parallelism exploitation (fine, medium, and coarse)*
- *Take advantage of heterogeneous HW, application workloads, and data types*
- *Develop efficient resource management mechanisms to favor locality and minimize data movement*

Solving the Heterogeneous Many-Core challenge: SPARTA



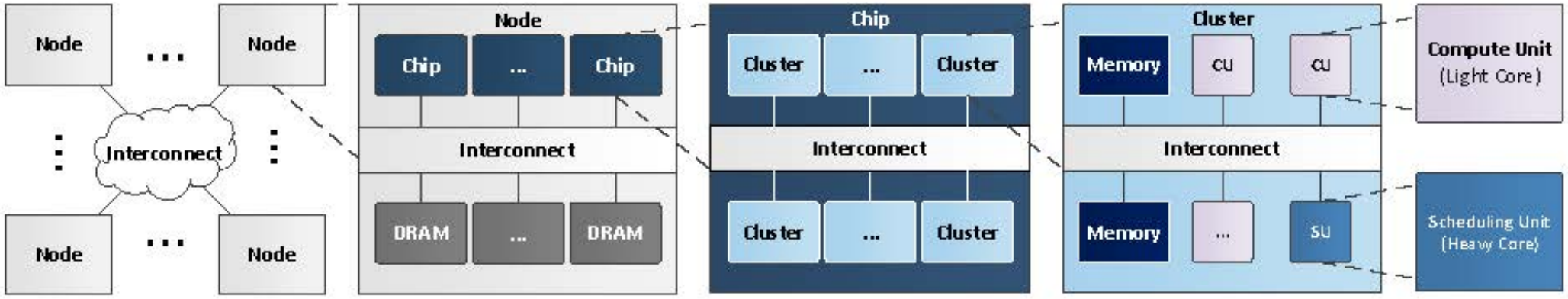
Codelet Model

- Codelet Definition:
  - *A codelet is a sequence of machine instructions which act as an **atomically**-scheduled unit of computation*
- Codelet Properties
  - Event-driven
  - Communication only through its inputs and outputs
  - Non-preemptive (with very specific exception)
  - Requires all data and code to be Local
- Codelet Fire Rules
  - Consume input token
  - Perform the operations within the *codelet*
  - Produce a token on each of his output



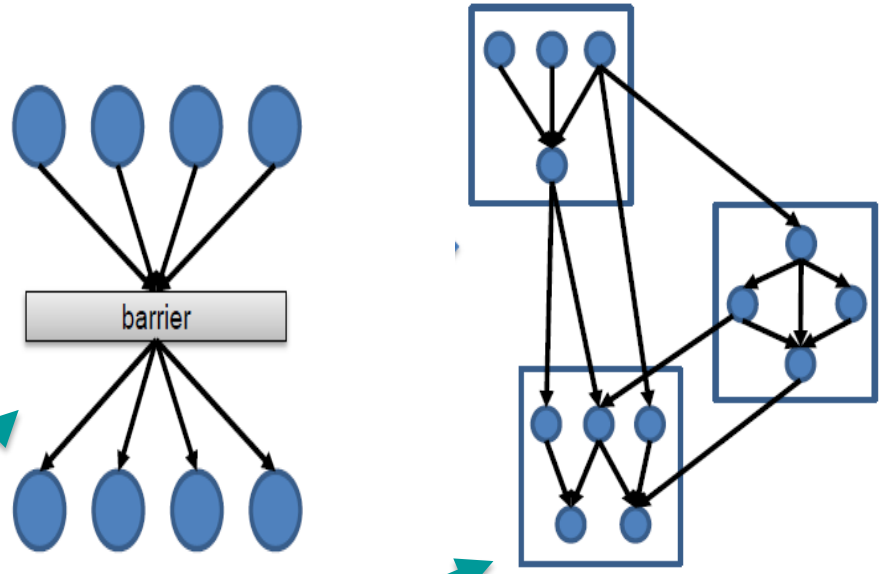
Codelet Abstract Machine (CAM) & Run-Time System (DARTS)

- CAM is a general purpose many-core architecture
  - Scheduling Unit
  - Computation Unit
- Map CAM to underlying hardware
- DARTS (Delaware Adaptive Run-Time System)
  - Invoke threaded procedures and map them on a given cluster of cores
  - Run the codelets contained within thread procedures.



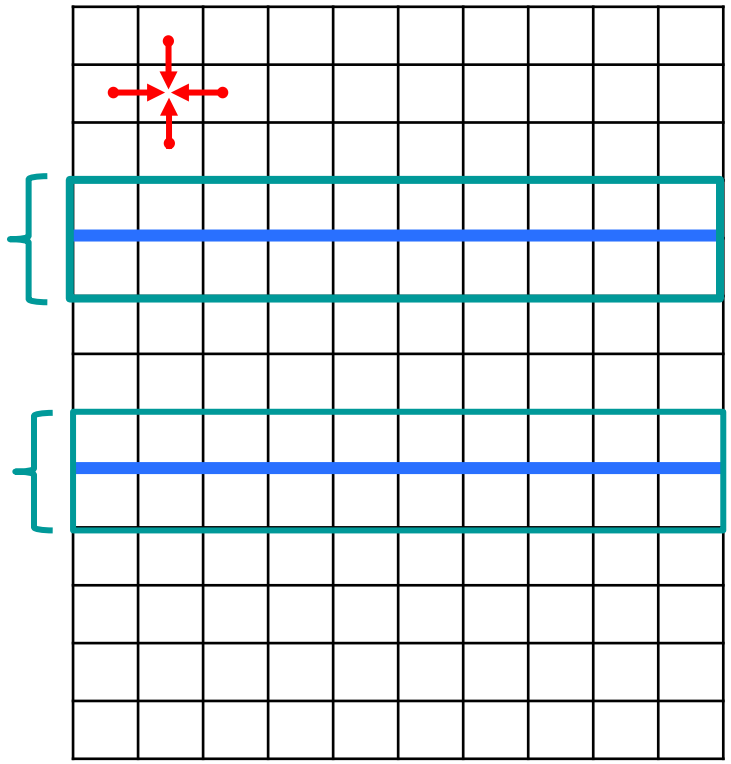
Multi-grain parallelism

- Platform
  - Many-core computing system
  - Shared-memory
- Two types of workload (applications)
  - CPU bound
  - Memory bound
- Parallelism
  - CPU bound
    - Coarse grain multi-threading model
  - Memory bound
    - Fine grain multi-threading model
  - Hybrid grain multi-threading model



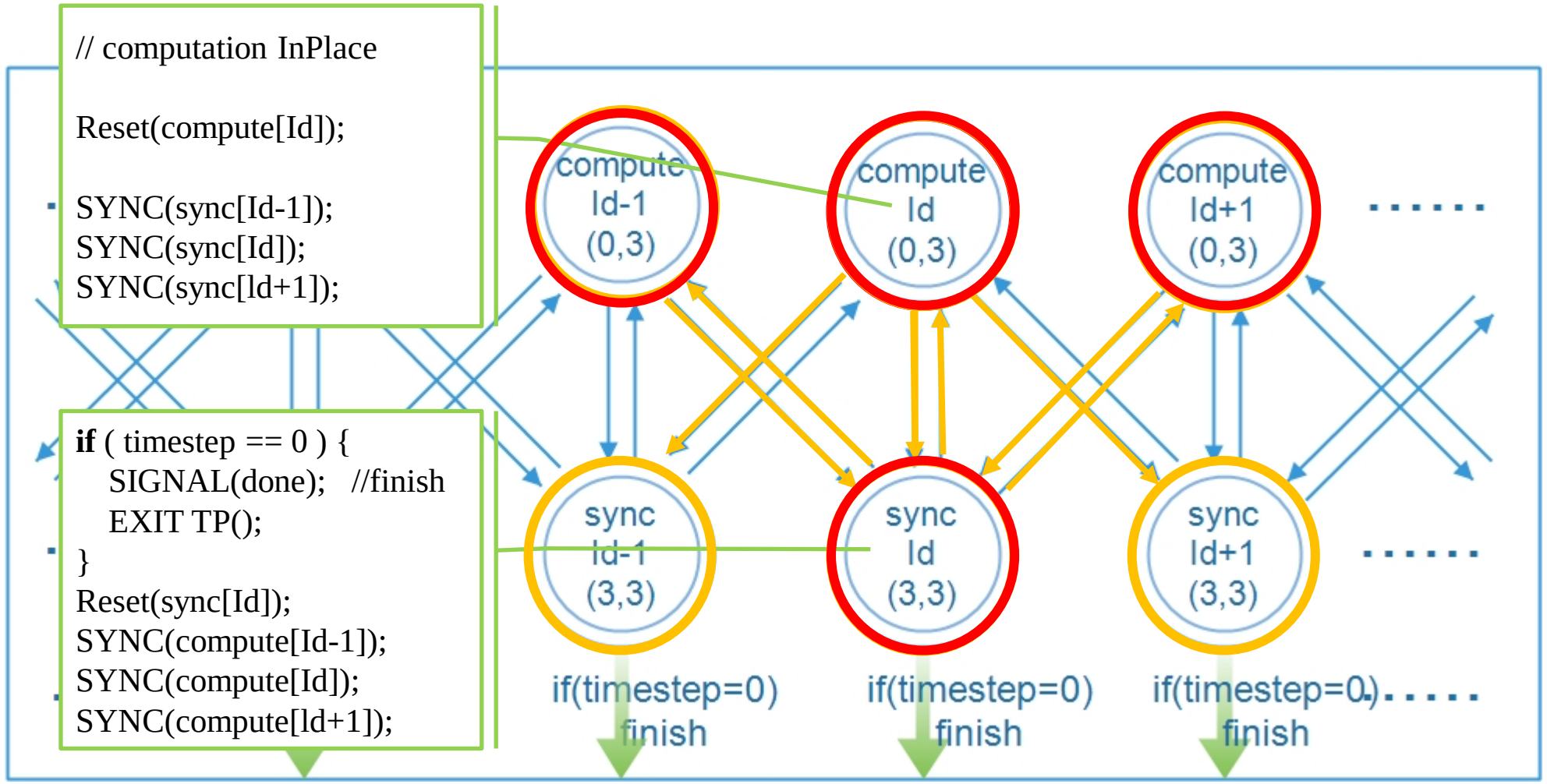
Stencil-based iterative computation

- **Stencil codes** are a class of iterative kernel which update array element according to some fixed pattern, called a stencil

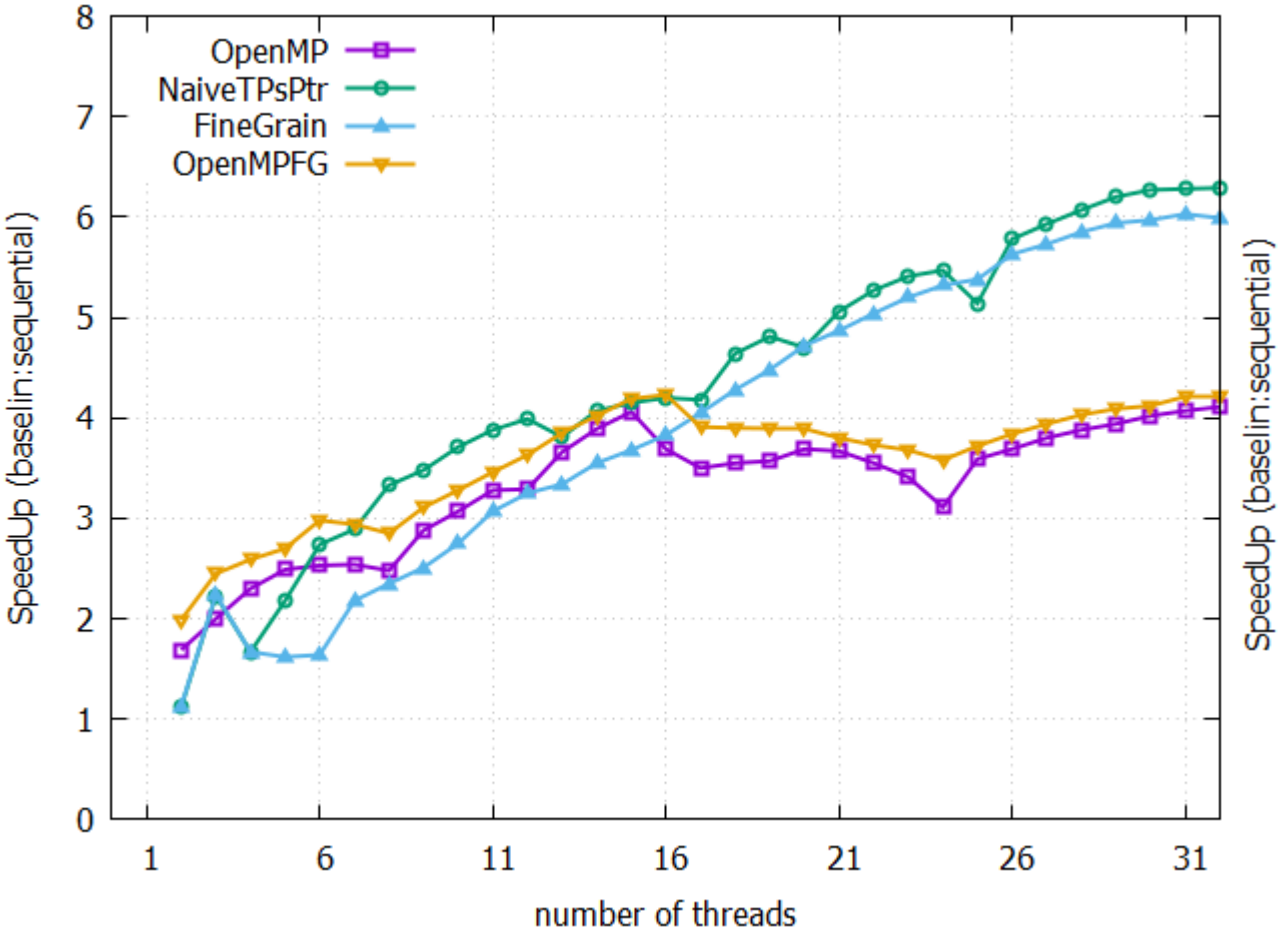


```
Core # computation: (InPlace)
upper, center, lower <- new double[n_cols-1];
Memcpy current<- shared,  lower<- SRC[lo];
while (--timestep >0){
for ( size_t i = lo; i < hi-1; ++i)
for (size_t i = 0; i <n_rows-1; ++i)
    Memcpy upper <- center;
    for ( size_t j = 1; j < n_cols-1; ++j)
    Memcpy center <- lower;
        DST[i][j]=(SRC[i-1][j]+SRC[i+1][j]+SRC[i][j-1]+SRC[i][j+1]) / 4;
    Memcpy lower  <- SRC[i+1]
} SWAP(&DST, &SRC);
    for ( size_t j = 1; j < n_cols-1; ++j)
        SRC[i][j]=(upper[i-1][j]+lower[i+1][j]+current[i][j-1]+current[i][j+1])/4;
}
SWAP(&DST,&SRC);
If (--timestep >0) {
SWAP(&DST,&SRC);
    Call Core # computation
If (--timestep >0) {
}
    Call Core # computation (InPlace)
}
```

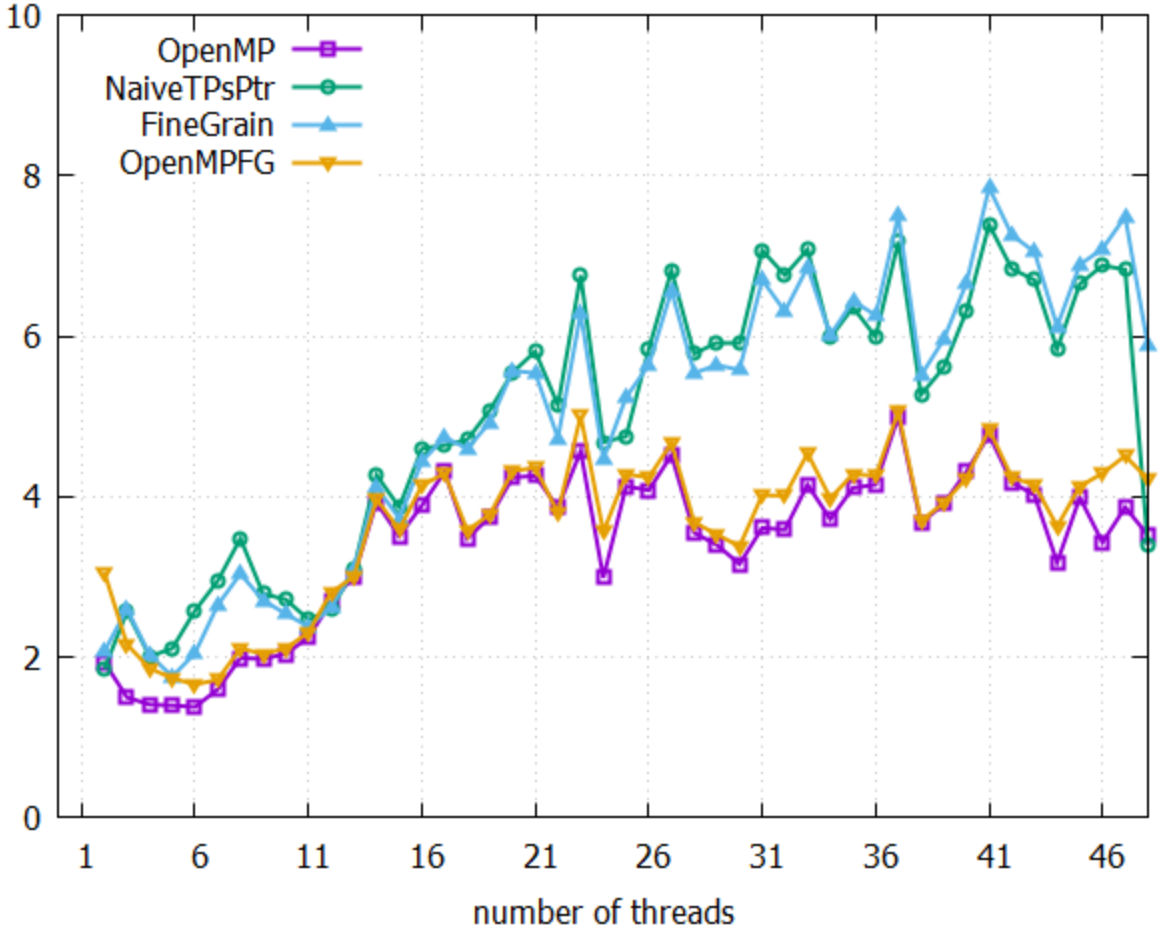
2D Stencil Graph –FineGrain/InPlace in 1 cluster



Strong Scaling



Intel: matrix size: 5000x5000

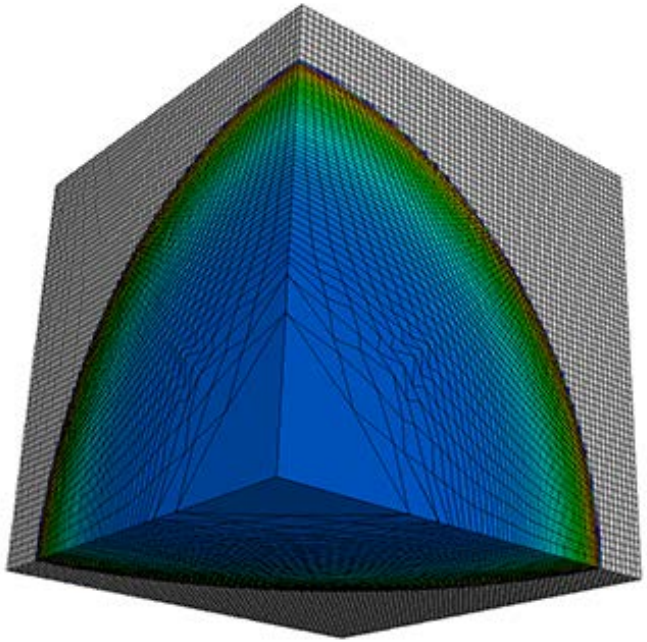


AMD: matrix size: 5000x5000

LULESH

--Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics

- LULESH is a hexahedral mesh-based physics code with two centering and time simulation constraints
  - Nodal centering
    - at the corners of hexahedra intersect
    - stores kinematics values, such as positions and velocities.
  - Element centering
    - at the center of each hexahedron
    - stores thermodynamic variables, such as energy and pressure
  - Time constraints
    - Limit how far in time the simulation advances at the next time step



Synchronization Granularity

- In a dependence-heavy kernel
  - Even in (data and control) regular codes
    - hierarchical fine/medium-grain synchronization is preferable to coarse-grain syncs (Barriers) for current multi/many core systems

- We obtained speedups up to 3.5x for 2D stencil and up to 1.35x for LULESH compared to OpenMP (official version)

Challenges for implementing scientific applications on heterogeneous System

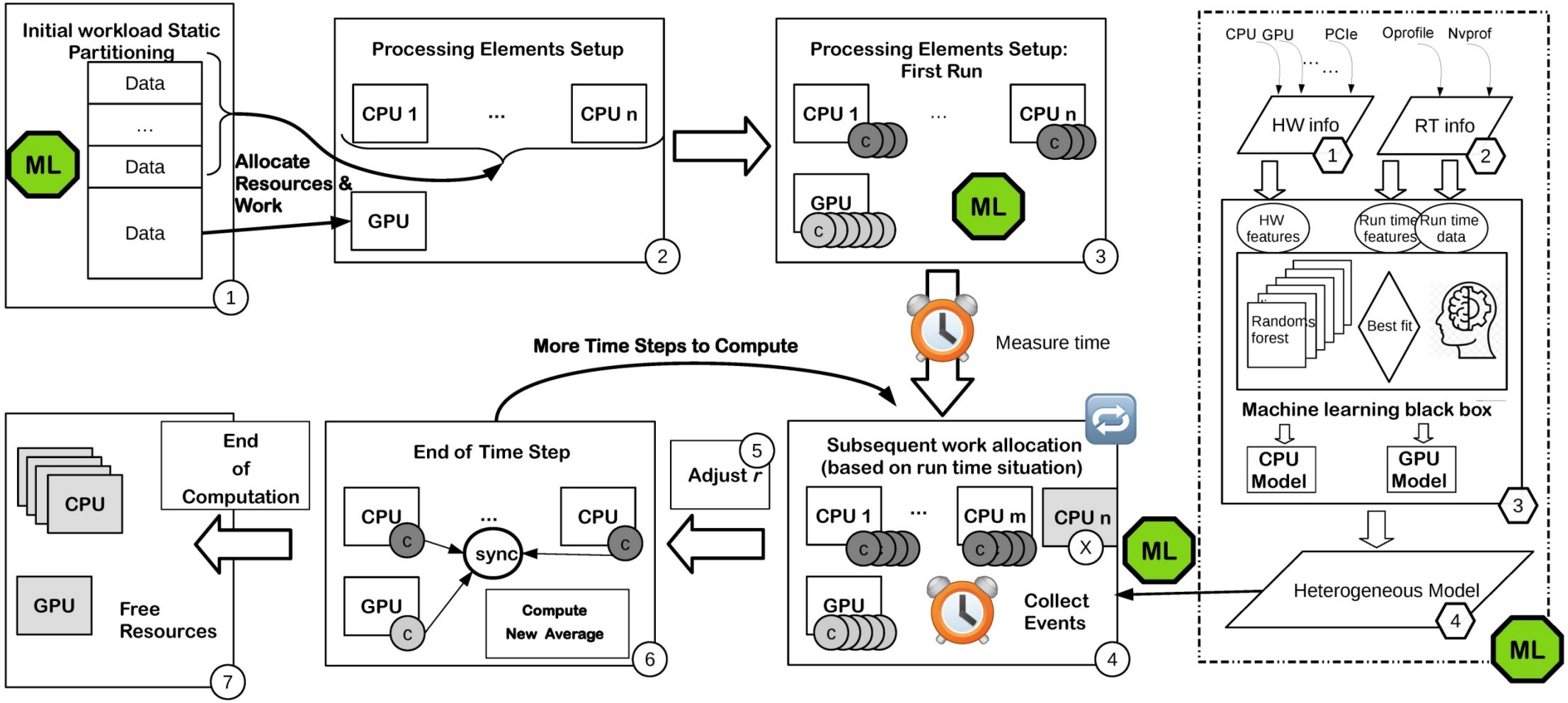
- Two popular ways
  - Fully offload the most compute-intensive parts of a given application to GPU(s)
  - Statically partition the compute-intensive parts between GPU and CPU

- The path less traveled: hybrid CPU/GPU computations
  - Requires a scheduler able to decide, online, which part of the workload to allocate, on which hardware resource
  - Must be able to adapt to dynamic variations in execution time over heterogeneous compute units
  - A mathematical model would be too complex to apply
    - Instead, rely on machine learning techniques (linear regression, random forest search, neural networks)

Our approach

- Combining online scheduling with Machine Learning to leverage load-balancing techniques in order to obtain the best workload partition between CPUs and GPUs.
  - An offline machine learning approach is employed to build the heterogeneous resources performance-workload (communication-computation) estimation model based on the analysis of the performance of pure CPUs and GPU.
  - The online scheduling adaptively adjusts the workload allocation based on performance model and the run time situation *(e.g.*, temporary unavailability of some devices because of power limitations).
  - Combining online and offline can improve flexibility and accuracy

Dynamic Adaptive Work Load (DAWL) Scheduling algorithm coupled with Machine Learning (IDAWL)



Dynamic Adaptive Work Load (DAWL) Scheduling algorithm coupled with Machine Learning (IDAWL)

- DAWL: Dynamic Adaptive Work-Load (DAWL) scheduling
  - Choose suitable computing resources (CPU or GPU, initial workload)
    - Estimate computing time on CPUs and GPUs using mathematical model
    - Initialize CPU/GPU configuration information
  - Run initial workload on chosen Cores
  - Adjust (dynamically) workload based on real time situation, *e.g.* , temporary unavailability of some devices because of power limitations

- Problems:
  - Mathematical model too complicated and low accuracy
  - Need to adjust model even with small HW configuration changes

Dynamic Adaptive Work Load (DAWL) scheduling algorithm coupled with Machine Learning (IDAWL)

- IDAWL: Profile-based Machine Learning Estimation Model For Iterative DAWL (IDAWL)
  - Collect HW information, *e.g.*, number of cores, number of socket, cache size, *etc*.
  - Collect application's profile information at runtime on pure CPU (using *oprofile*) and pure GPU (using *nvprof*)
  - Cluster algorithm to group features
  - Build profile-based estimation model
    - Choose best fit model from regression, random forest, SVM, *etc*. algorithm to build estimation model
  - Obtain the impact factor of parameter
  - Build hybrid model and inject information to DAWL corresponding stages

Conclusions

- Challenges for High Performance Computing
  - Core count increases dramatically per chip
  - For performance and energy/power savings reasons, systems are heterogeneous
  - Traditional coarse-grain approach to parallel computing is not sufficient anymore

- Event/data driven parallel computing for HPC was shown to be a viable solution to tackle such challenges, I presented three contributions in this context:
  - Synchronization granularity on many-core shared-memory systems
  - Workload balance on heterogeneous many-core systems
  - Data flow movement and resources allocation for stream processing

Ongoing Work

- Build a communication ML Model to estimate communication cost among more heterogenous computing resources.
  - Communication between CPU and multiples GPUs
  - Communication between CPU and FPGA
  - Communication among CPU, GPUs, and FPGA

- Integrate more ML models to IDAWL, such as neural network and online ML algorithms.

- Augment our model with power consumption parameters to enrich IDAWL and determine good trade-offs between performance and power on heterogeneous architectures

Thanks, any questions?