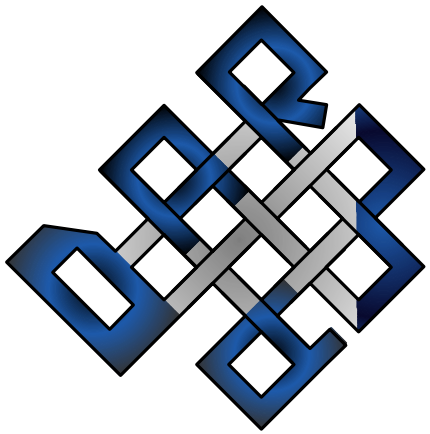
SAND2017-9781C



Exploring DARMA Abstraction Layer for PIC and DSMC Kernels on Next Generation Platforms

Aram H. Markosyan, Matthew Bettencourt, Janine C. Bennett, Jonathan Lifflander, David S. Hollman, Jeremiah Wilke, Hemanth Kolla, Chris Moore, Robert L. Clay (PM)

DSMC 2017, Santa Fe

August 30, 2017

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

SAND Number: SAND2017-9086 C

Outline

- AMT + DARMA Overview
- Introduction to SimplePIC
- Performance Results
- Conclusions and Future Work

Extreme-scale HPC system architectures introduce a number of complexities

Sandia National Laboratories

- Performance Heterogeneity
  - Accelerators
  - Thermal throttling
  - General system noise
  - Responses to transient failures
- Energy Constraints
- Decreased system reliability
- Deep memory hierarchies

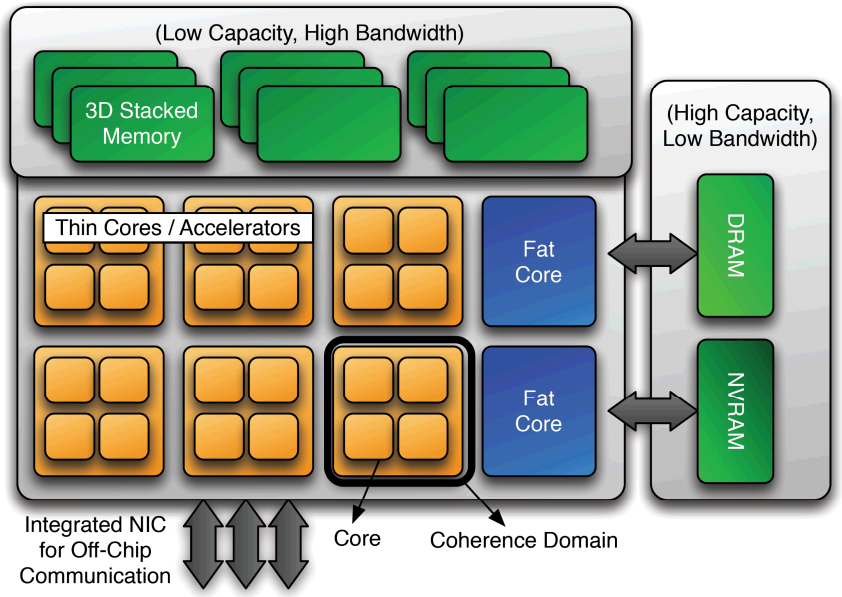


Image courtesy of www.cal-design.org

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

Current imperative programming models and runtime systems require mitigation of challenges largely at application-developer level

AMT research is focused on mitigating system complexities at the runtime system-level

- Abstractions provide a separation of concerns
- Removal of system-level specifics from application code
- Task parallelism
- Asynchrony, overlap of communication and computation
- Load balancing

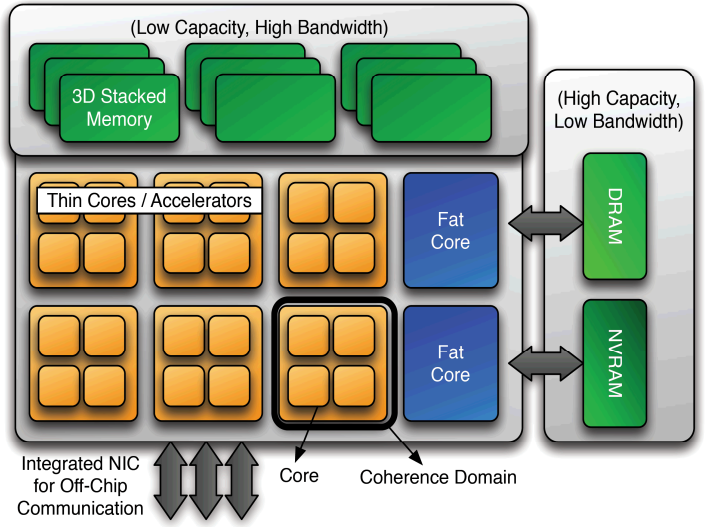


Image courtesy of www.cal-design.org

COMPUTER ARCHITECTURE LABORATORY
EXASCALE DESIGN SPACE EXPLORATION

AMT models are a shift from an ***imperative to declarative*** programming paradigm

Imperative vs declarative programming: a simple example

Sandia National Laboratories

Imperative

```
Get a piece of bread
If likes mustard
    Add mustard
If not vegetarian
    Add meat
Add cheese
Add veggies
Put more bread on top
Cut in half
```

Programmer uses explicit statements to control program state and prescribe order of operations

Declarative

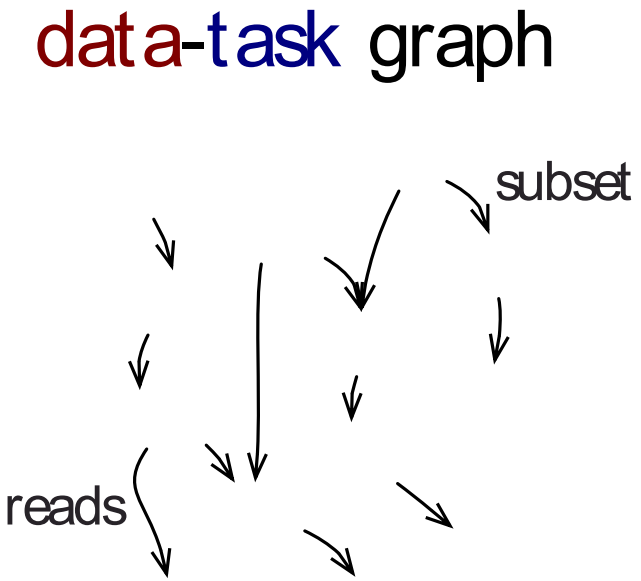
```
Make me a sandwich
```

Programmer expresses logic without prescribing control-flow

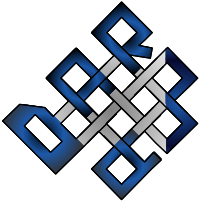
Asynchronous many task (AMT) models and runtime systems provide a declarative programming approach

Sandia National Laboratories

- Directed acyclic graph (DAG) encodes data-task dependencies
  - Enables coarse-grained, distributed memory analog of instruction-level parallelism
    - Data prefetching
    - Out-of-order task execution based on runtime dependency analysis
  - DAG can be annotated to capture
    - Tasks' read/write usage of data
    - Task needs a subset of data
  - Additional information enables runtime to reason more completely about
    - When and where to execute a task
    - Whether to load balance



- Existing runtimes leverage DAGs with varying degrees of annotation



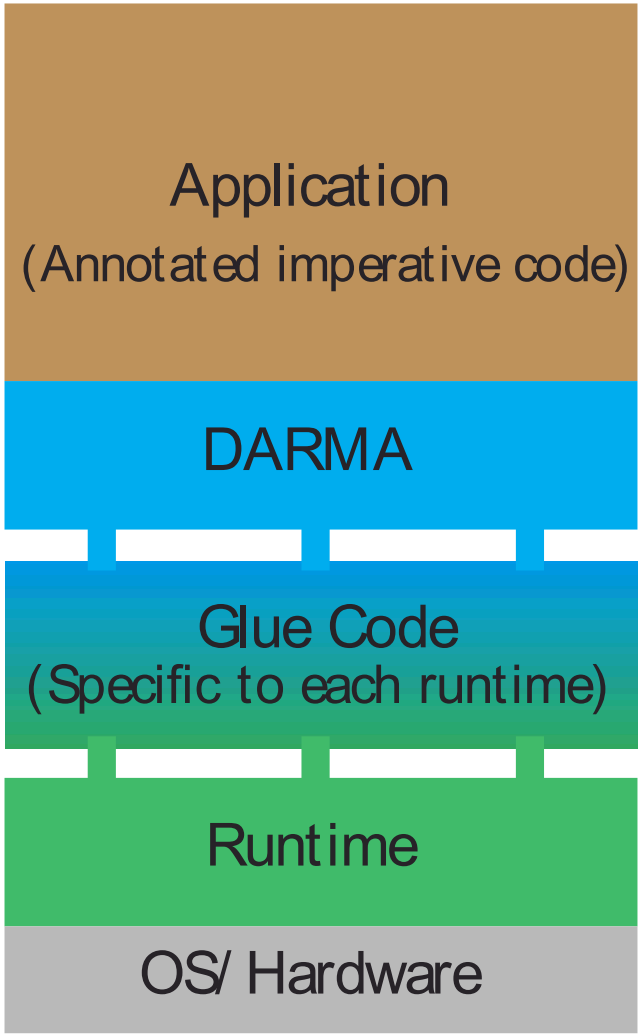
What is DARMA?

Sandia National Laboratories

DARMA is a C++ abstraction layer for asynchronous many-task (AMT) runtimes.

It provides a set of abstractions to facilitate the expression of tasking that map to a variety of underlying AMT runtime system technologies.

How does DARMA simplify the shift from imperative to declarative programming?

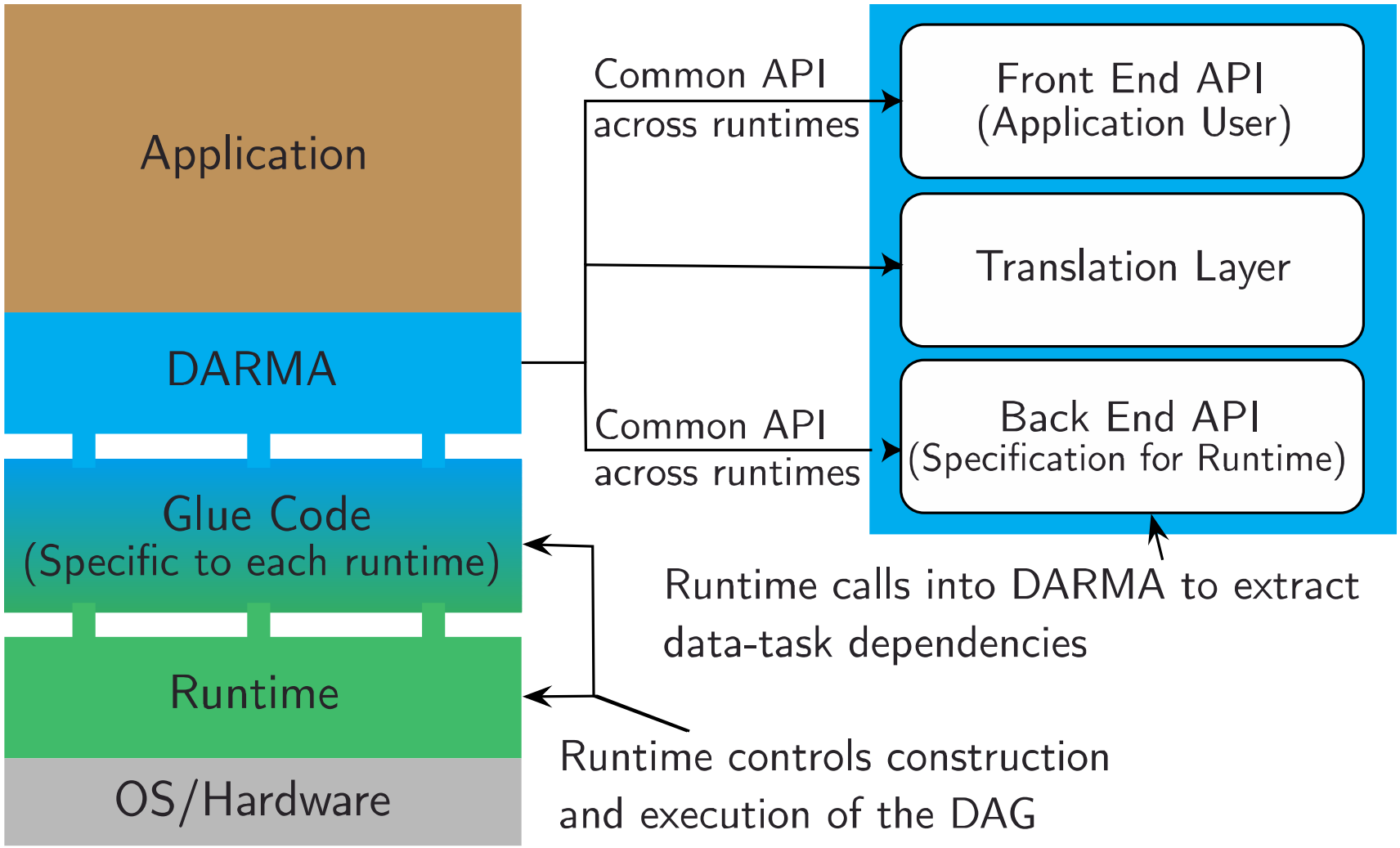


The application "produces" work. Annotated imperative code is processed by DARMA, which builds the DAG incrementally at run-time.

The DAG is generalization of a producer-consumer work queue

The runtime system is in charge of control-flow and the order in which it "consumes" tasks off of the DAG.

What is DARMA?



EMPIRE: ElectroMagnetic Plasma In Radiation Environments

Sandia National Laboratories

- SNL is developing a new code base for plasma simulations
- Component based approach using the Trilinos framework
- The PIC component of Empire is the basis for our proxy app work
- Two sets on unknowns, mesh data and particles
  - Domain decomposition on the fields and the particles can be out of balance, (e.g. particle collision work is out of balance)
  - Calculations are localized so colocation is important
  - Work can be created in one location and migrate to a different location
- Potential solution – over decomposition
  - Over decomposition breaks the problem up into more units than you have computational cores
  - Load balance at a middle level of work
  - Overlap computation and communication

From MiniPIC and SimplePIC to EMPIRE

Sandia National Laboratories

- MiniPIC is an electrostatic PIC miniapp build on MPI+Kokkos.
- A proxy app SimplePIC was developed
  - Particle move kernel from MiniPIC on a structured mesh built on DARMA.
  - 3D with various boundary conditions
  - MPI based version of SimplePIC was developed for benchmark purposes.
- The current code design flow is: SimplePIC → MiniPIC → EMPIRE.

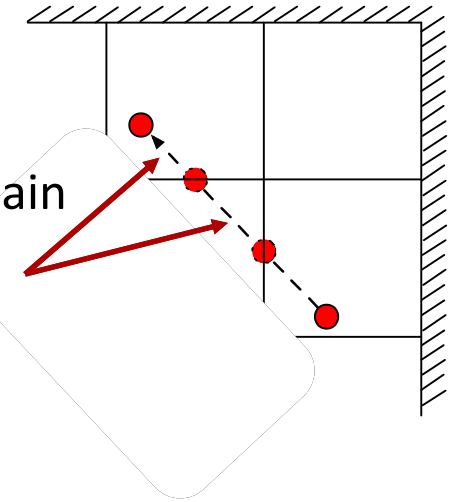
- SimplePIC and the DARMA backend were built up together
  - Every single new and experimental feature of DARMA was first tested on the SimplePIC (performance/productivity feedback)
  - Made DARMA a more performant, productive, feature rich and robust programming model

SimplePIC Proxy Overview

- SimplePIC includes only particle move kernel
  - Push particles in constant applied field
  - Does not solve Poisson eqn. – this is integrated into MiniPIC
- Domain Decomposition: 2-level 3D structured grid
  - $P_x \times P_y \times P_z$ grid of boxes (patches), $n_x \times n_y \times n_z$ grid within each box
- Computational costs:
  - $O(N_{particle})$ computation (memory bound), $O(N_{particle} \times patch_{surf}/patch_{vol})$ communication,
- Proxy goal: serve as test ground for PIC algorithm design and development on DARMA

SimplePIC Proxy Algorithm

- Decompose problem into patches and assign them to processing units
- For every patch initialize the swarm (particles on that patch)
- For each time step do (**iteration**)
  - For each particle in the swarm do
    - Advance particle until it reaches the patch interface or time expires
    - If time is not expired do
      - Put particle in the migrants (a buffer, corresponding to that patch interface)
      - Remove particle from swarm
  - Compute the total number of migrants in the entire domain
  - While total number of migrants > 0 do (**micro-iterations**)
    - For every patch interface exchange the migrants
    - For each interface do
      - For each particle in migrants do
        - Advance particle until it reaches the patch interface or time expires
        - If time expired add particle to swarm, otherwise put in migrants
    - Compute the total number of migrants



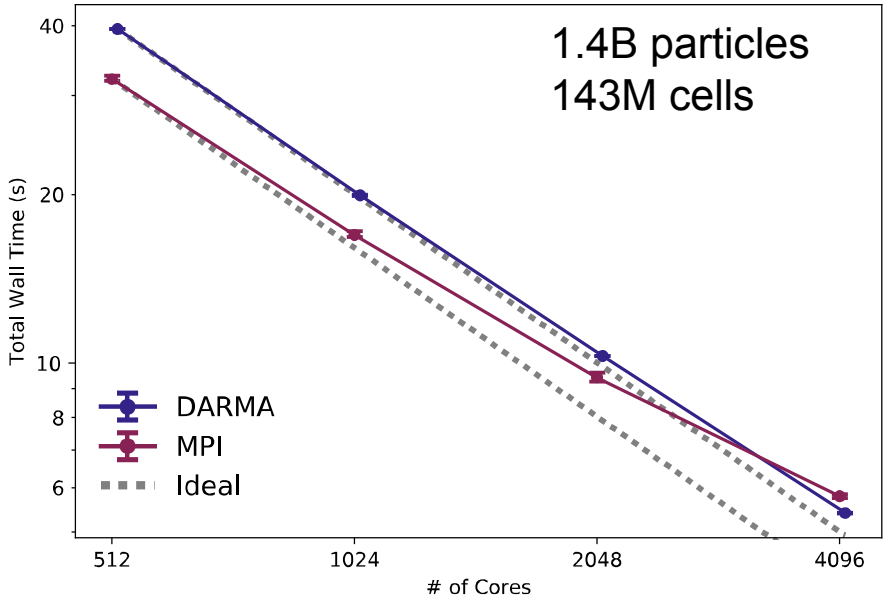
Balanced and Unbalanced SimplePIC Studies

- Balanced use case assesses overheads with respect to MPI-only implementation
  - Every computational cell has N randomly placed particles (5 - 30), with random velocities (|v| = const).
- Imbalanced use case assesses benefits of overdecomposition and load balancing in DARMA
  - Initially place 80% of particles into the 20% of the domain creating load imbalance in the system.
  - The computational experiment was designed such that the system will reach to a fully balanced state in 500 iterations and come to the initial state in 1000 iterations.
- In all studies we kept CFL number to a value of 0.96, which translates into at most 3 micro-iterations per time step.

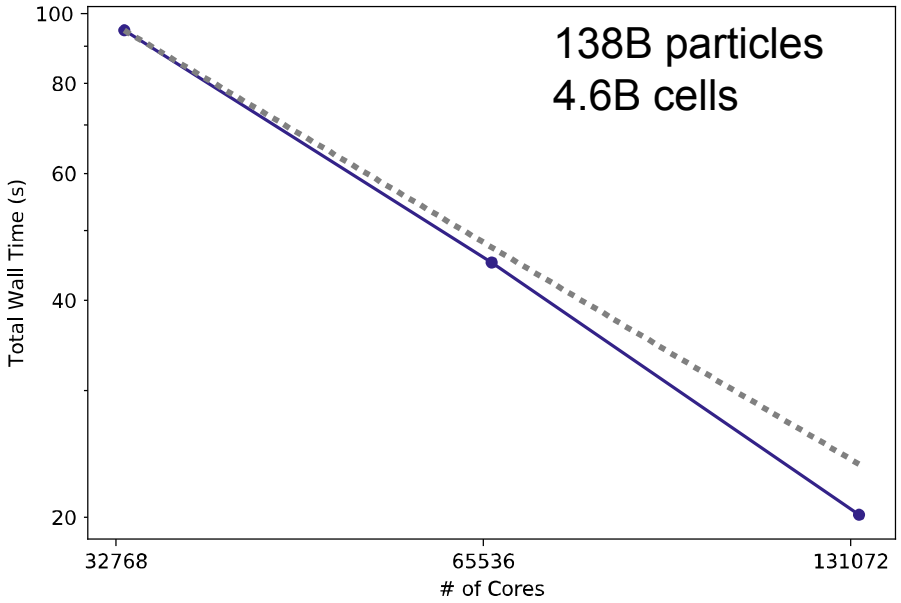
Strong scaling of balanced SimplePIC up to 131K cores/2K nodes (KNL)

Sandia National Laboratories

Mutrino (KNL, 4K cores)



Trinity (KNL, 131K cores)



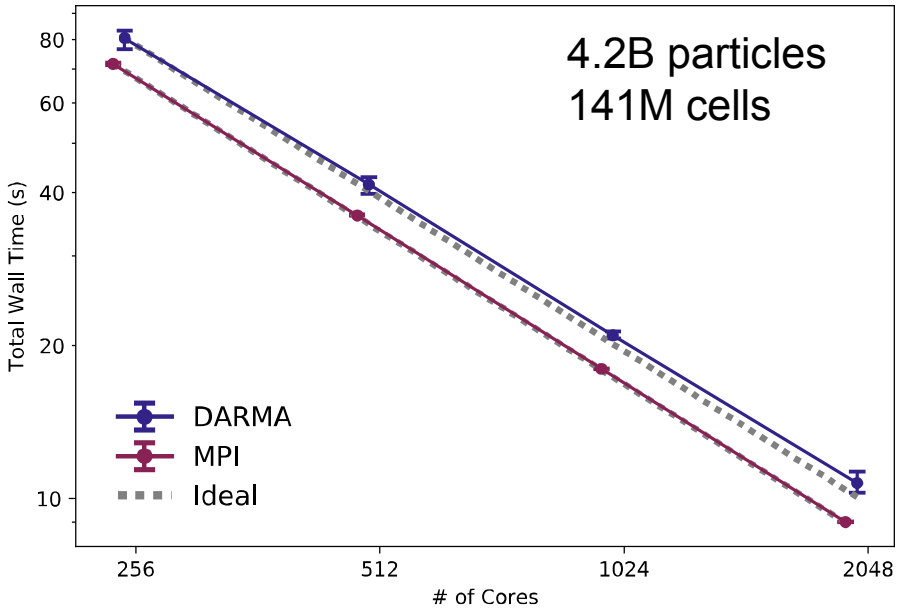
- DARMA overhead with respect to MPI is -5-24%.
- On 4K cores, grain size is too small and, hence, degraded scaling.
- MPI scaling degradation is likely due to MPI only launch on KNL.

- DARMA scales super-linearly up to 131K cores.

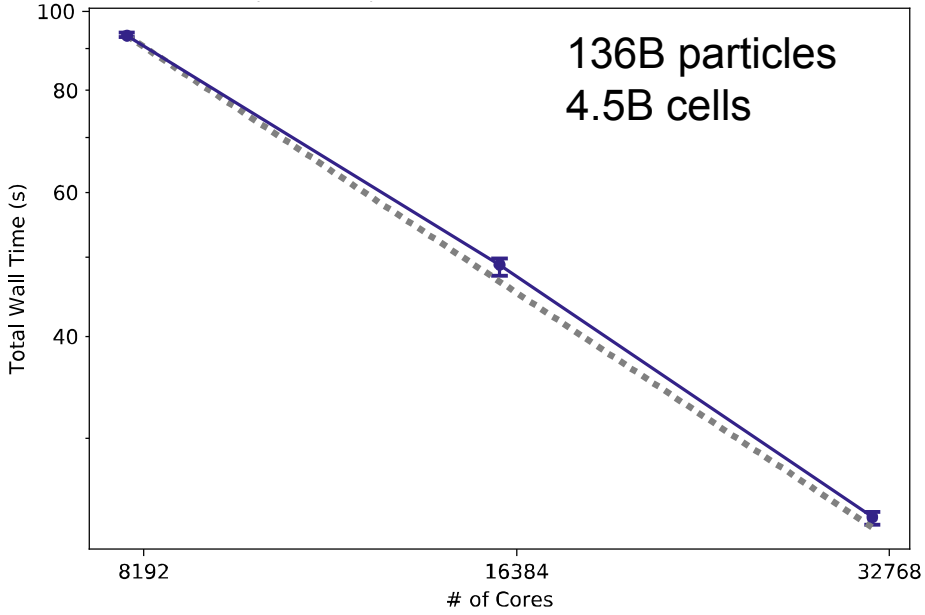
Strong scaling of balanced SimplePIC up to 32K cores/2K nodes (Haswell)

Sandia National Laboratories

Mutrino (Haswell, 2K cores)



Trinity (Haswell, 32K cores)



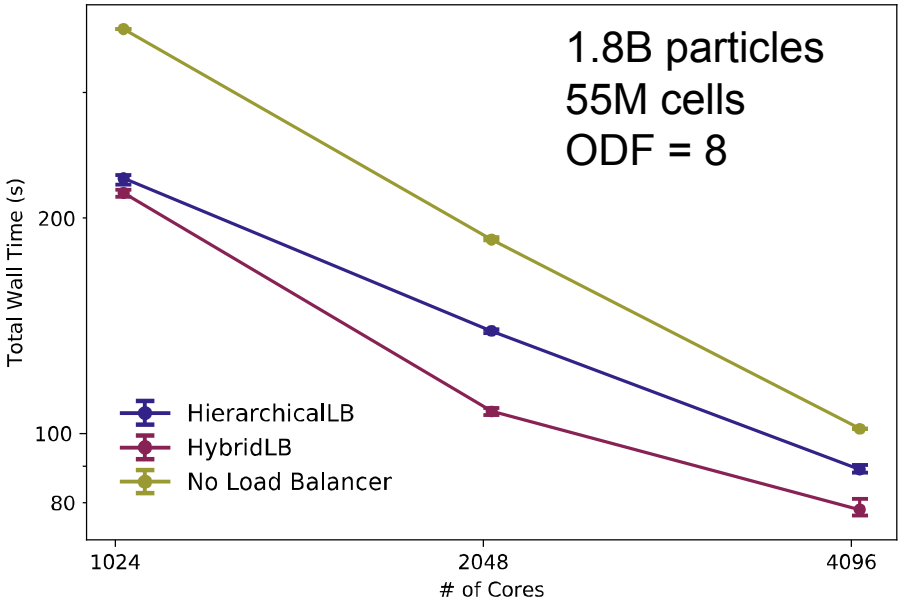
- DARMA overhead with respect to MPI is 12-19%.
- On 2K cores, grain size is too small and, hence, DARMA does not have perfect linear scaling.
- MPI scales ideally on up to 2K.

- DARMA scales consistently good on up to 32K cores.
- Slight overheads can be explained by the small problem size on higher core counts.

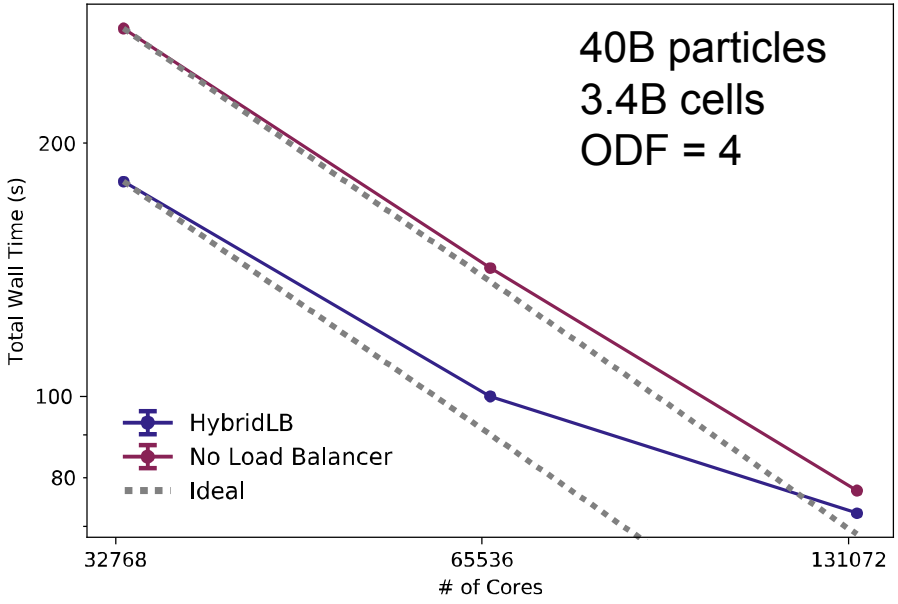
Strong scaling of imbalanced SimplePIC up to 131K cores/2K nodes (KNL)

Sandia National Laboratories

Mutrino (KNL, 4K cores)



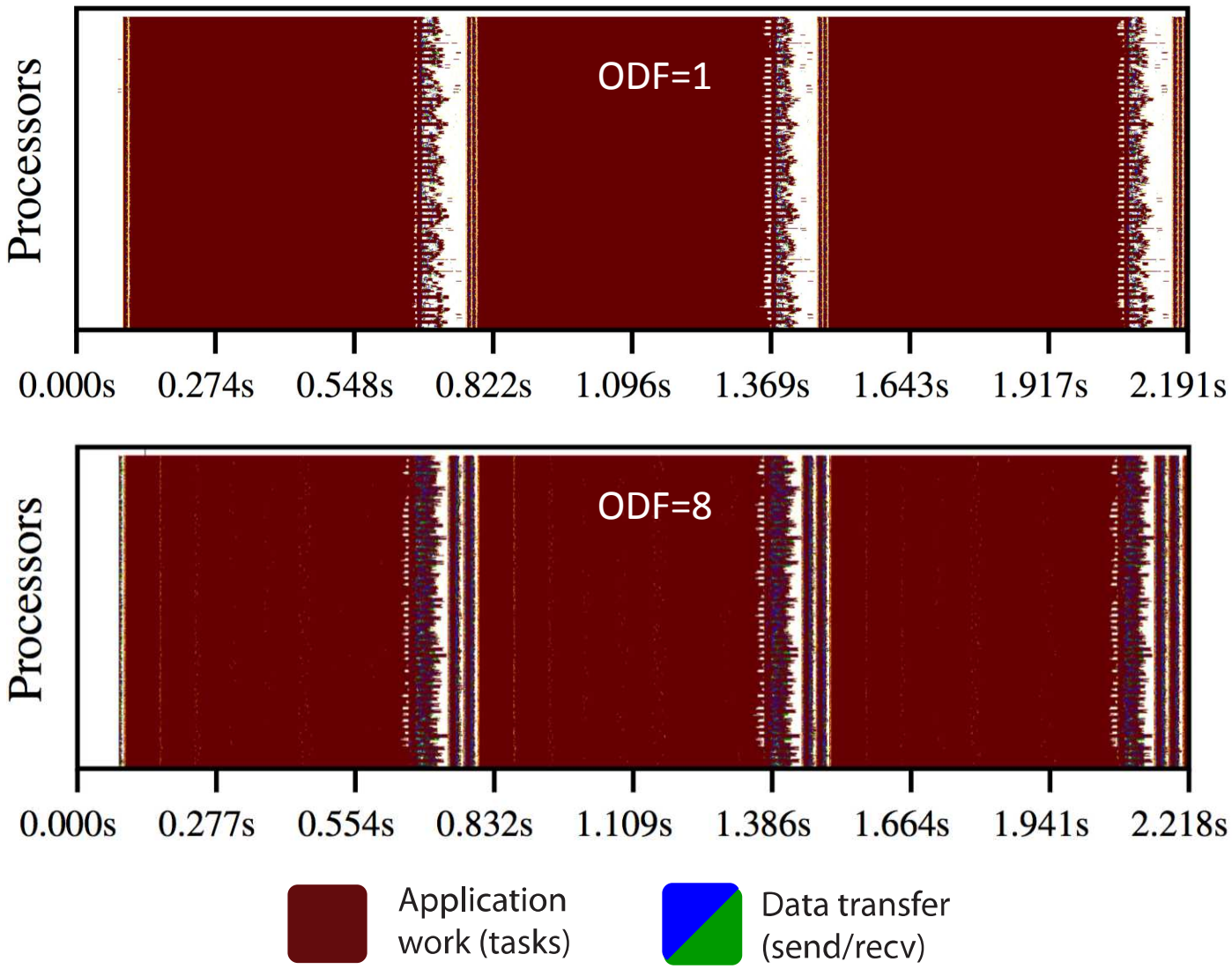
Trinity (KNL, 131K cores)



- For lower core counts, load balancing provides around 50% speedup.
- For higher core counts, at least at this overdecomposition level, speed up due to a load balancer is 20%.
- These trends are similar for Haswell.

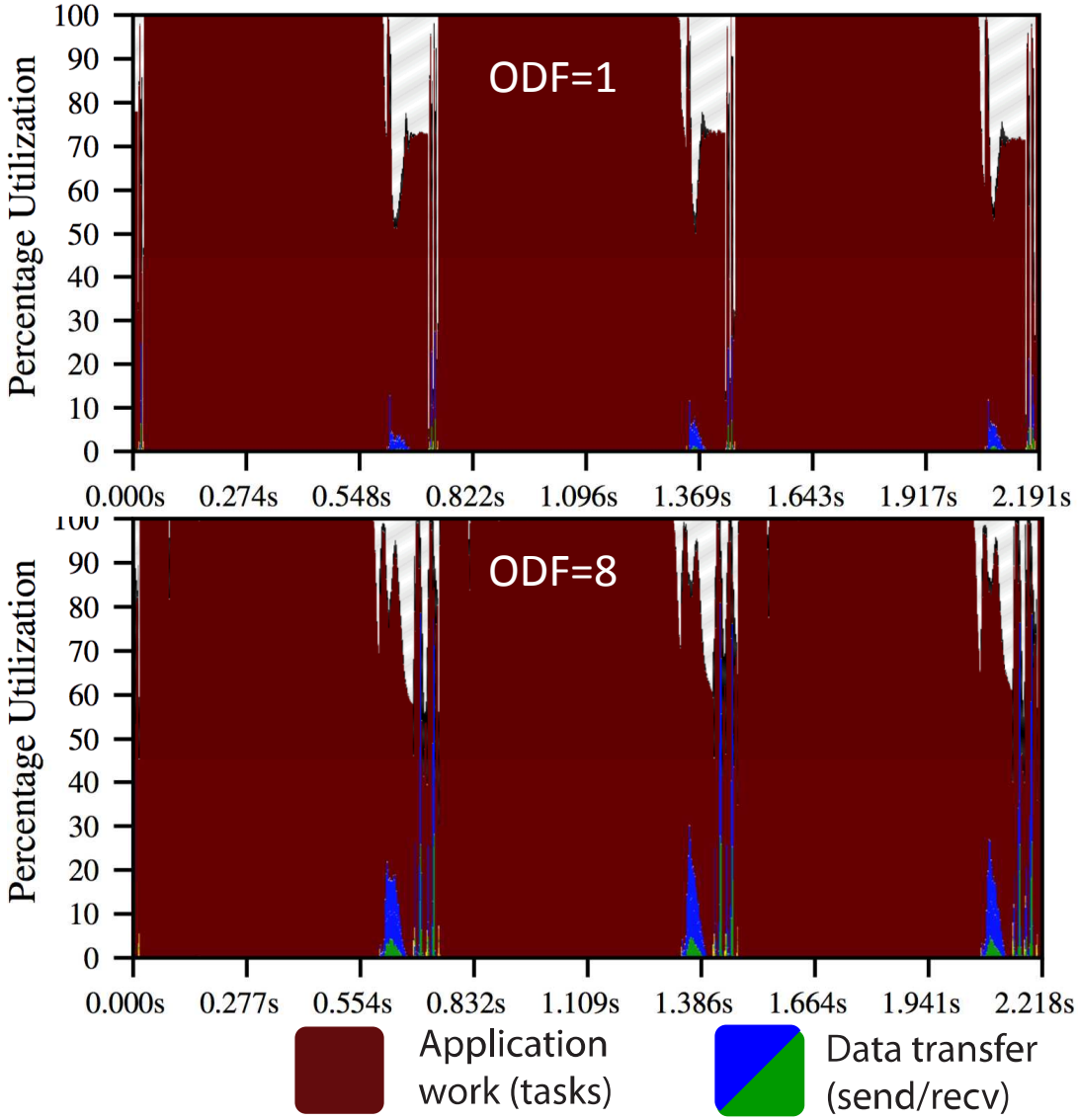
- Similar trends are present on Trinity at these higher scales.

Time Profile Graph of Balanced SimplePIC for DARMA on 2k Cores/64 nodes (Haswell) for 3 Iterations



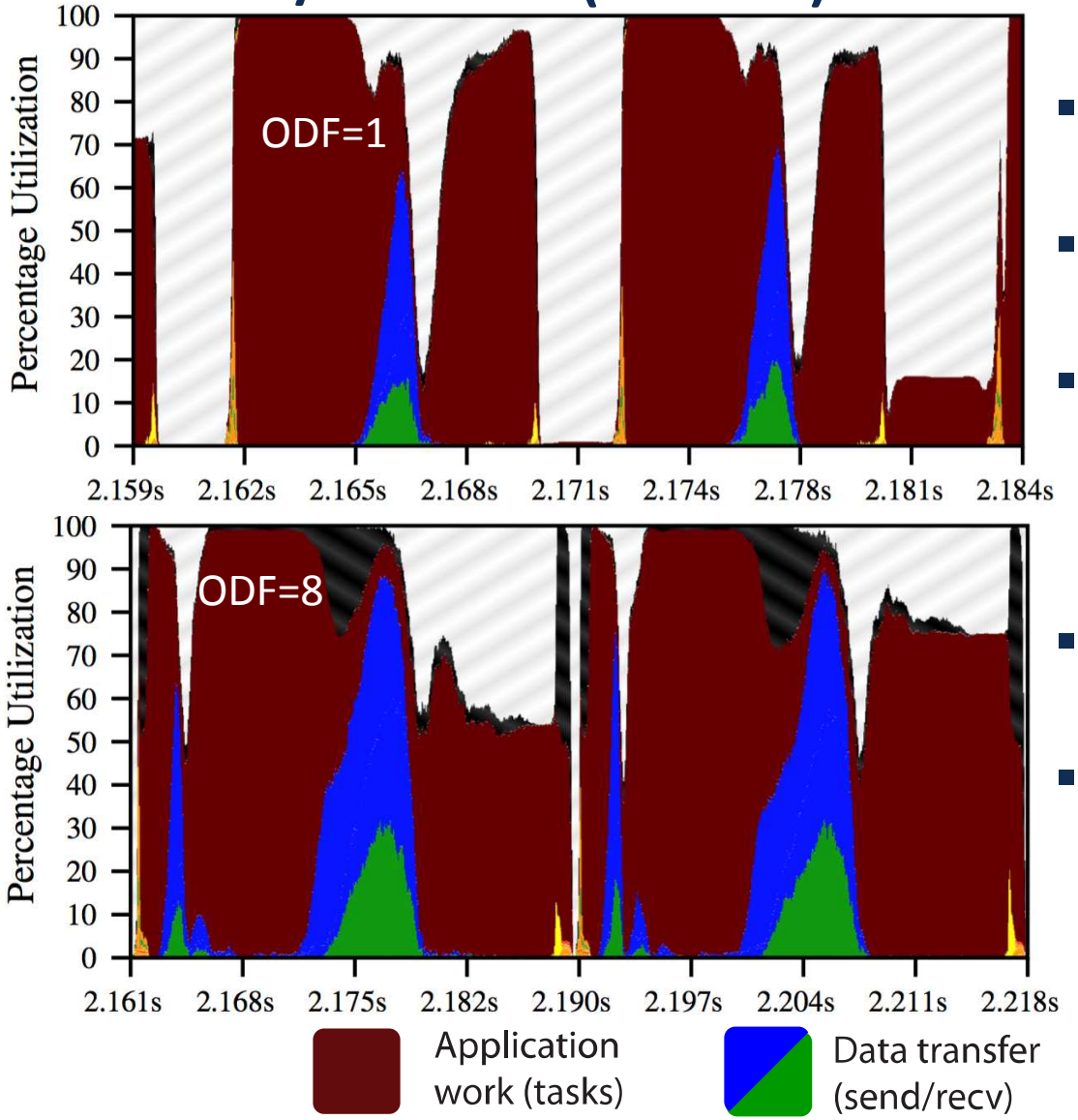
- x-axis is time and y-axis are different cores
- Most of the time is spent executing application tasks
- There is a small amount of idle time (white) at the end of each iteration

Percentage Utilization Graph of Balanced SimplePIC for DARMA on 2k Cores/64 nodes (Haswell) for 3 Iterations



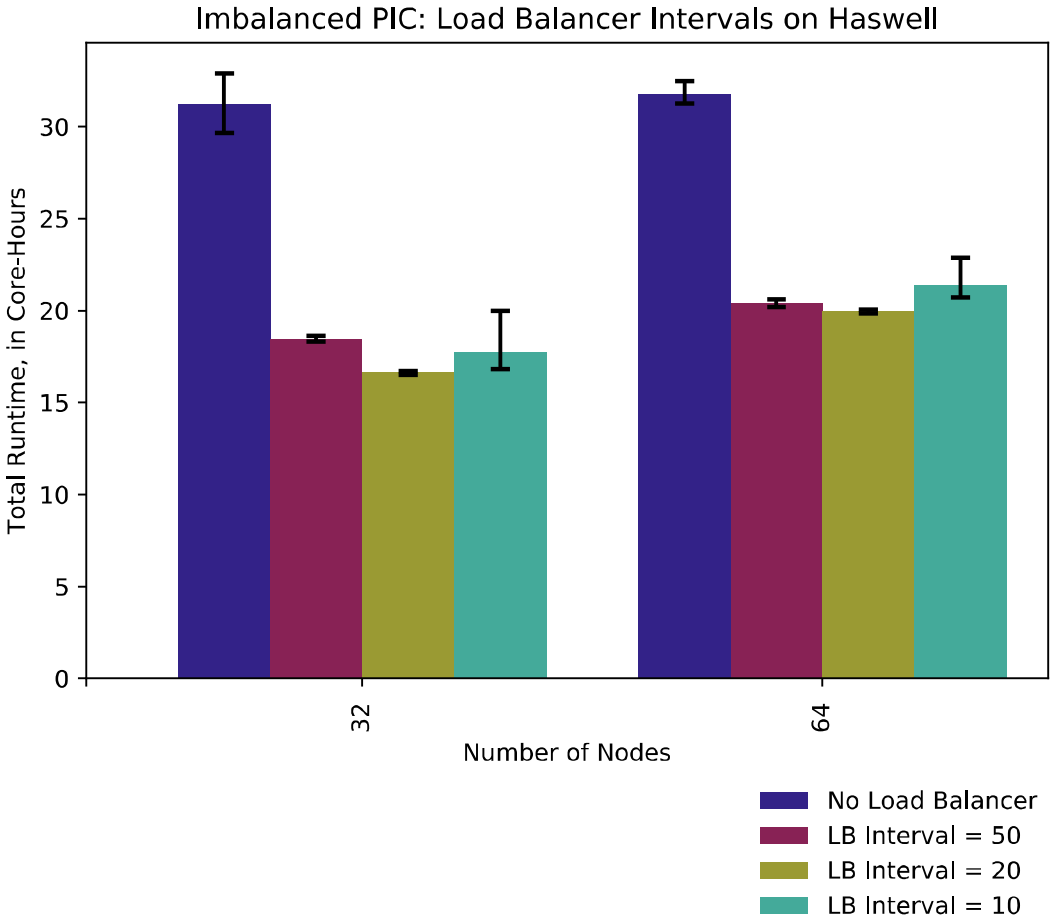
- x-axis is time and y-axis is the proportional aggregate of work type spent across the worker cores
- With an overdecomposition factor of 8 (ODF=8) the data transfer time is slightly increased
- The idle time at the end of the iteration is slightly reduced with ODF=8 because the system is able to overlap communication with computation

Time Profile Graph of Balanced SimplePIC for DARMA on 2k Cores/64 nodes (Haswell) for last 2 micro iterations



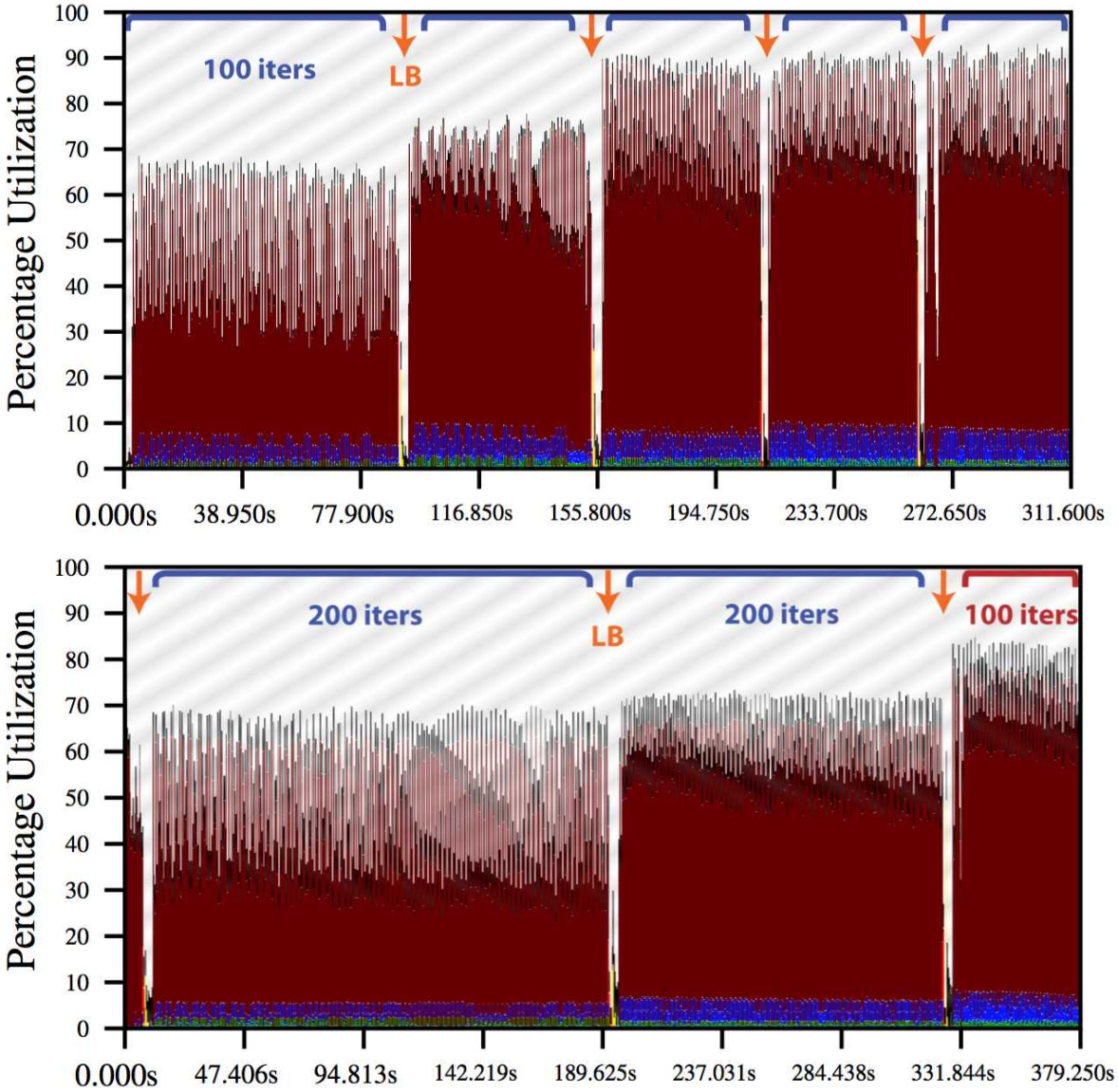
- Processor utilization for 2 micro iterations
- Note the scale: this is 25 milliseconds
- Overdecomposition increases the execution time because data transfer is increased (note the increase in green and blue area)
- More particles must cross the boundaries with smaller boxes
- Overall processor utilization is increased because there is more overlap with communication

Load Balancing Frequency of SimplePIC for DARMA on 2k Cores/64 nodes (Haswell)



- Calling load balancing only once improves the performance almost 2x.
- The optimal load balancing frequency for this particular case is 2 times.
- In general, optimal frequency depends on factors like the cost of load balancer, the grain size, overdecomposition factor.

Projection views of imbalanced SimplePIC for DARMA on 2K cores (Haswell)



- Significant improvement in load imbalance with more frequent calls to load balancer.
- The overhead (cost) of load balancer is essentially constant.
- Over 2x CPU utilization increase after the first load balancer call (in both cases).

Conclusions on SimplePIC Performance Study

Sandia National Laboratories

- Balanced SimplePIC study stressed DARMA overheads with respect to MPI. In the worst cases we are off by 25%.
- Balanced SimplePIC also showed an excellent scalability on 131K cores.
- Imbalanced SimplePIC demonstrated the benefits of overdecomposition and load balancing on 131k cores, while maintaining strong scalability.
- Addition of DSMC kernel will help increasing the grain size and do more computation and communication overlap.

Lessons learned on productivity for SimplePIC proxy

Sandia National Laboratories

- Manual (dynamic) overdecompositon and load balancing in MPI can be very tedious and error prone task even for structured PIC. For unstructured case, the situation is very complex.
- Data decomposition in DARMA provides intuitive mechanisms for work load balancing, while runtime handles scheduling.
- DARMA abstractions are fairly intuitive and provide a productive environment for code design and development.

From SimplePIC to MiniPIC (and to EMPIRE)

Sandia National Laboratories

- As designed, SimplePIC served as a test ground for a algorithmic exploration for MiniPIC (EMPIRE).
- MiniPIC was further simplified (Kokkos and MPI dependences were removed) and move kernel was DARMA-tized.
- DARMA-tization of the DSMC kernel is in progress.

- The prerequisites for DARMA to move forward (towards EMPIRE code base) are: *Kokkos and MPI interoperability*

Future Work

- Focus on DARMA
  - Interoperability
  - Hardening/Tuning
  - Productivity tools (timers, performance profilers, debugging aides)
  - Devops, documentation, and testing
- Focus on SimplePIC and MiniPIC
  - Incorporate a collide kernel in SimplePIC
  - DARMAtize MiniPIC completely