Brown Deer Technology

DRC HIGH PERFORMANCE TECHNOLOGIES GROUP

UNIVERSITY OF DELAWARE

US ARMY RDECOM

U.S. Army Research, Development and Engineering Command

Investigation of Parallel Programmability and Performance of a Calxeda ARM Server Using OpenCL

ARL

TECHNOLOGY DRIVEN. WARFIGHTER FOCUSED.

David Richie
Brown Deer Technology

Lori Pollock
University of Delaware

James Ross, Jordan Ruloff
Dynamics Research Corp.

Song Park, Dale Shires
Computational Sciences Division
U.S. Army Research Laboratory

UnConventional High Performance Computing 2013 (UCHPC 2013)

Outline

- Hardware
  - ARM multi-core CPUs
  - Calxeda ARM server
- Software Stack
  - OpenCL, CLRPC, STDCL
  - Distributed task parallel programming model
- Configurations tested
- Benchmarks
- Conclusion and future work

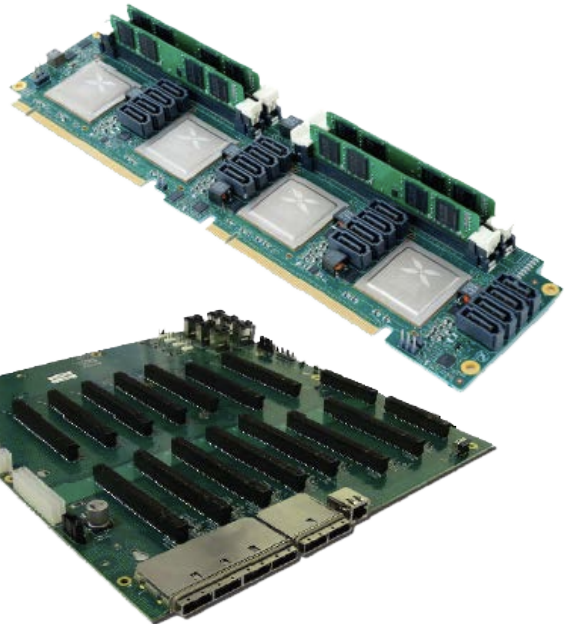
ARM CPU Architecture

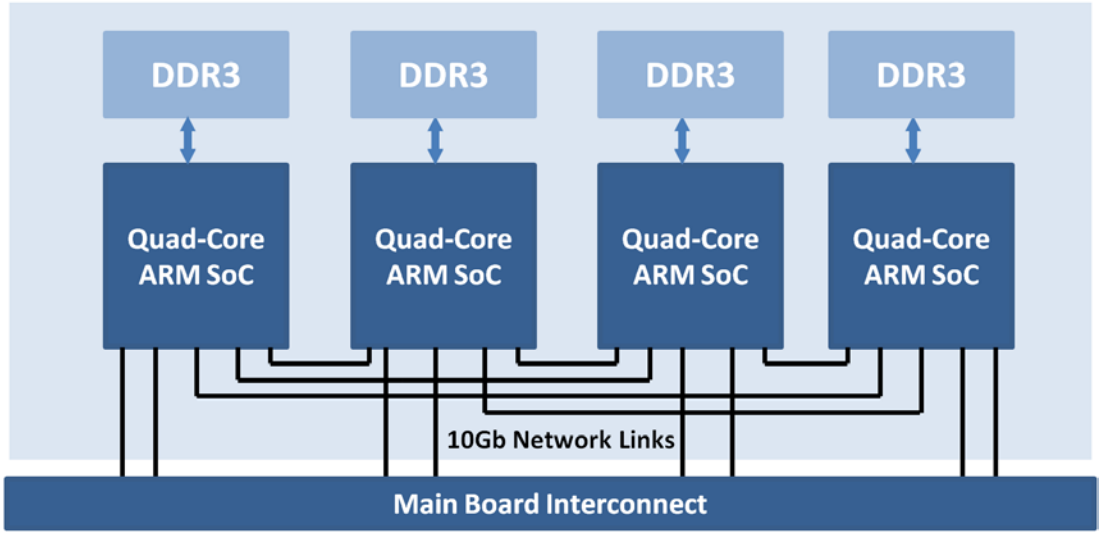
- ARM is a family of RISC CPUs targeting low-power and embedded markets.
- ARM Holdings licenses the IP core to third parties
- You likely have one in your pocket. They appear in about 95% of smartphones

Enterprise ARM Platforms	Enterprise x86 Platforms
“RISC” - reduced complexity ISA	“CISC” – specialized ISA
Power and Energy priority (<5 W)	Speed and Performance priority (~100W)
Lower clock speeds ~1-2 GHz	Higher clock speeds ~3 GHz
Generally four (4) cores per system	Eight (8) to 16 core systems
Low memory per system (up to 4 GB)	High memory systems (16-256 GB)
64-bit Vector Floating Point coprocessor	256-bit SSE and AVX instructions
Smaller die size (~5 mm^2)	Larger die size (~100 mm^2) adds cost

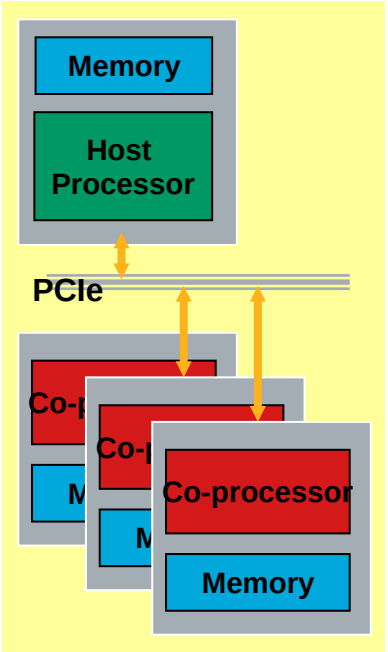
Calxeda ARM Server

- Fully populated 2U box contains 48 quad-core SoCs
  - 192 ARM Cortex-A9 cores @ 1.1-1.4 GHz
  - 192 GB DDR3 (1GB/core)
  - Five (5) 10 Gb network links per SoC
  - 900W PSU (usage typically < 300 W)
  - Up to four (4) SATA ports per SoC





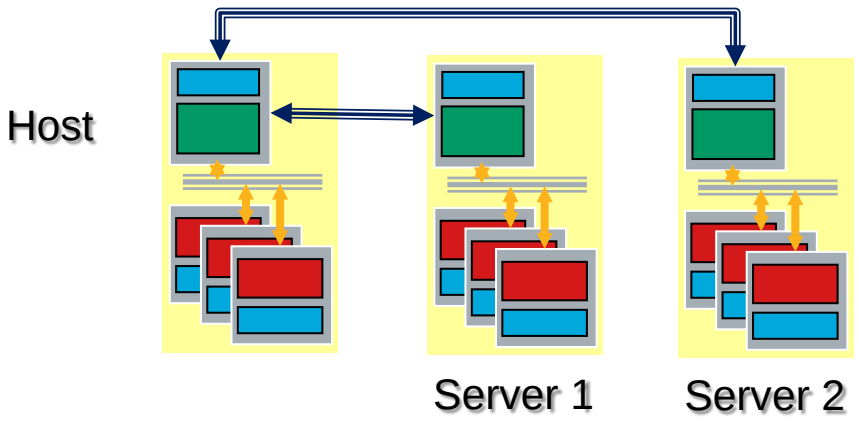
- OpenCL™ is an industry standard programming API
- Issues addressed by OpenCL
  - Distributed memory management
  - Concurrency of data movement and device execution
  - Platform/vendor portability of APIs
- OpenCL provides ...
  - Platform and runtime layer for managing concurrent execution of operations across multiple devices
  - C language extension for programming compute devices
  - Platform/device independent API with broad industry support



- Alternatives:
  - **MPI** – Data parallel, not task parallel. Runs on multi-core CPUs and clusters
  - **OpenMP** – Runs on Symmetric Multiprocessor (SMP), single node
  - **CUDA** – Similar to OpenCL. Nvidia GPUs only, single node.

CLRPC: Basic Idea

- The OpenCL API forces natural separation between high-level functionality (host code) and heavy compute tasks (kernel)
- OpenCL RPC (Remote Procedure Call) can extend the host code transparently to access the compute resources across networked platforms
- OpenCL RPC provides access to larger pool of OpenCL devices from single application
- This is the inspiration for CLRPC



CLRPC

- CLRPC: Remote Procedure Call (RPC) implementation of OpenCL
  - OpenCL capabilities of a node exported to the network
  - Transparent layer to developer
  - OpenCL code doesn’t need to be modified
  - Devices still separated by OpenCL platform concept
    - Programmer needs a separate OpenCL context per CLRPC server
  - Host able to execute OpenCL program even if it lacks any remote OpenCL implementation or compute device
- CLRPC Daemon (clrpcd) runs in background on a node and forwards CLRPC commands to local OpenCL library
  - Daisy chaining of networked nodes as a means to create tiered execution to possibly reduce bottlenecks

CLRPC: Server-Side

- Compute resources are exported using a CLRPC server (clrpcd)
- clrpcd is run in the background on each node, for example:

  clrpcd -a 192.168.1.36

- The server will listen for CLRPC requests on the default port, and perform all host API calls on behalf of the client
- On client-side the available OpenCL platforms can be augmented with one or more clrpcd servers identified by IP address and port

- FAQ: Does the server just have “wrappers” for the OpenCL API calls?
  - In a sense, yes, but there are non-trivial issues that must be resolved, mostly related to implicitly asynchronous behaviors expected from within the OpenCL client applications
  - A naive implementation that “blocked on everything” would be useless

STDCL

- STDCL™ (STandard Compute Layer) as an interface to OpenCL
- Lightweight UNIX-style C API interface for HPC
- Similar simplicity and features to CUDA, yet more portable
- Creates standard OpenCL contexts: *stdcpu*, *stdgpu*, *stddev*, *stdnpu*
- The context *stdnpu*, is unique in that it is a “super context” of networked OpenCL devices through CLRPC
- *stdnpu* context ignores platform details and presents to user all available devices across multiple platforms

STDCL

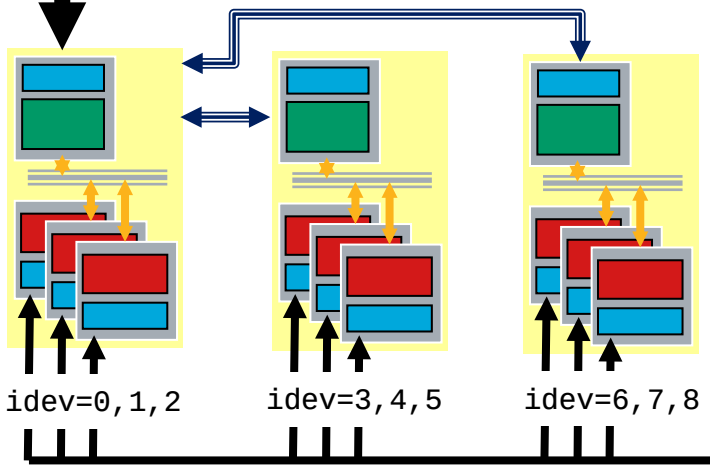
- Without STDCL *stdnpu*, directing devices requires management:
  - For each Node
    - For each OpenCL Platform
      - For each OpenCL Context
        - » For each OpenCL Command Queue
          - » For each OpenCL Device
- STDCL stdnpu abstracts these layers so code becomes:
  - For each OpenCL Device
- Reduces source code complexity significantly with very little overhead

- Concurrent execution of an OpenCL kernel on multiple networked devices

```
// Loop over all networked devices
for(idev = 0; idev < ndev; idev++)
   clforka(stdnpu, idev, krn, &ndr, CL_EVENT_NOWAIT, a, b, c);

// Wait on each device in turn
for(idev = 0; idev < ndev; idev++)
   clwait(stdnpu, idev, krn, 0);
```

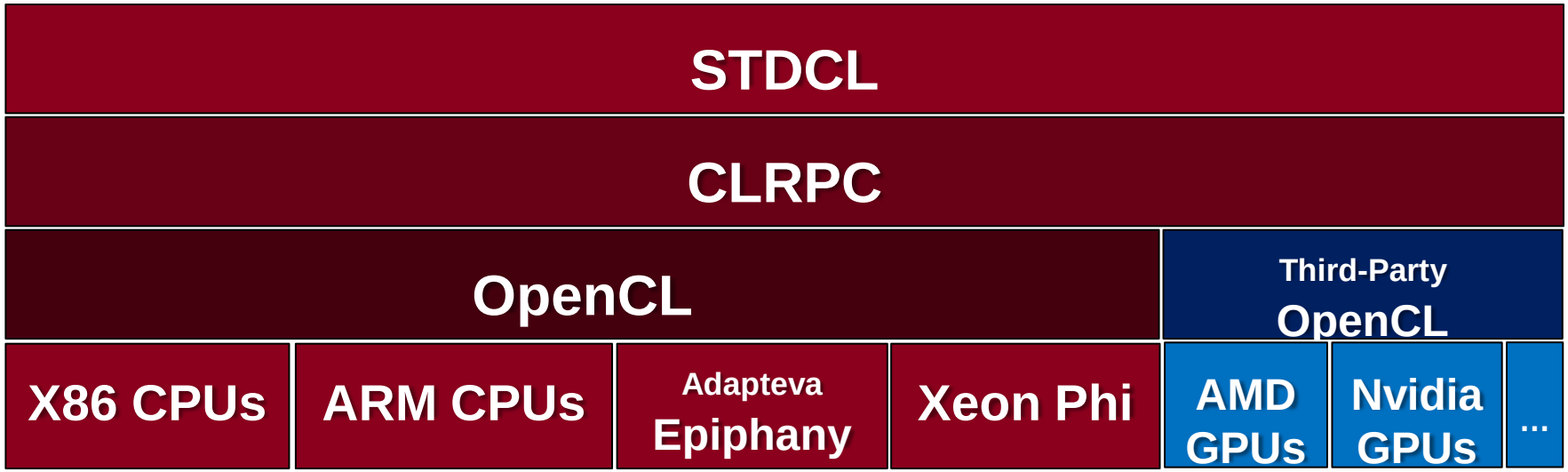
Host program runs on a single host



Compute kernel runs on each device *concurrently*

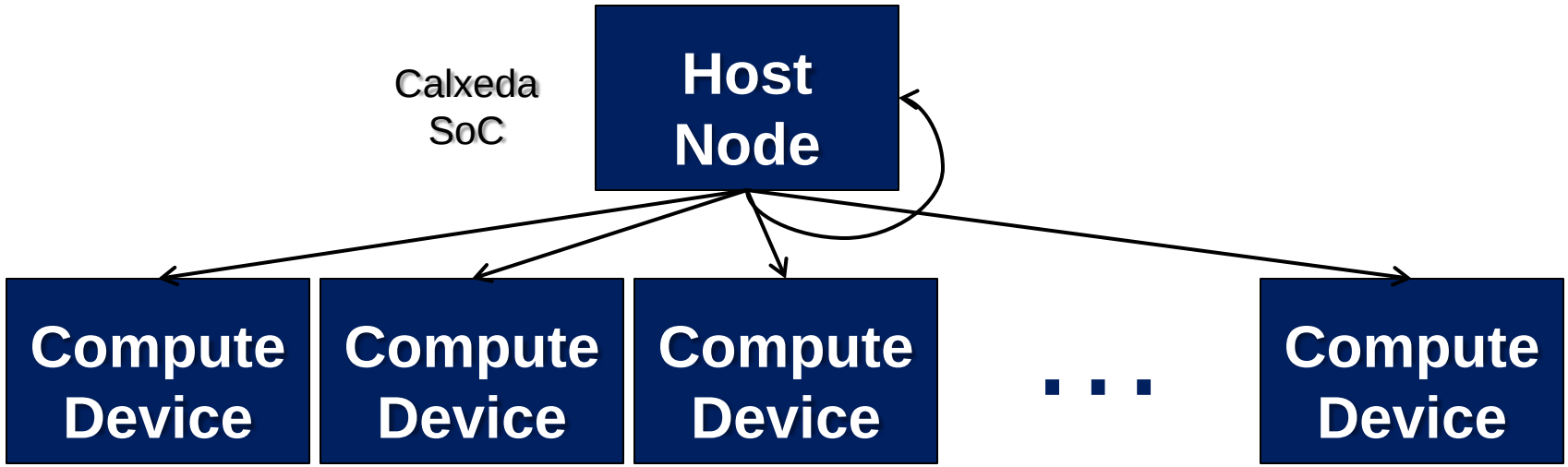
COPRTHR SDK Software Stack

- CO-PRocessing THReads™ (*COPRTHR™*) *SDK*
- Write STDCL code once, run on any architecture, multiple devices and nodes
- Includes OpenCL for x86 and ARM multi-core CPUs, Adapteva Epiphany RISC Arrays, and Xeon Phi (beta). ARM has no vendor-supplied implementation
- COPRTHR SDK allows for the most portable source code for our group
- Besides those listed above, we've investigated AMD, Nvidia, and Adreno GPUs
- Primary support for Linux/BSD and some support for Android and Windows



Programming Model

- We are investigating a task-parallel programming model for distributing work across a cluster of Calxeda nodes based on STDCL
- A dedicated host node distributes work via OpenCL/STDCL calls and treats networked nodes as OpenCL compute devices
- Host may also use it's own resources for compute

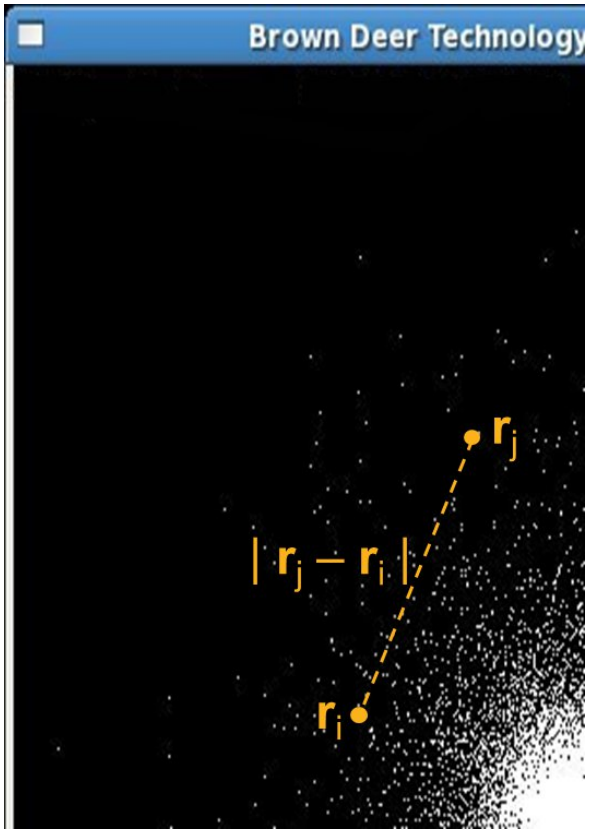


Configurations Tested

- Access to the Calxeda box was provided by Exxact Corporation
- At the time, five (5) quad-core nodes were available for use
  - Running Ubuntu 12.10
  - GCC 4.7 and common utilities for a Linux development platform
  - Includes baseboard management controller (BMC) utility for measuring and recording power of individual nodes within the system
- Tested configurations included:
  - OpenCL direct
  - CLRPC remote
  - CLRPC local (special case)
  - *stdnpu*

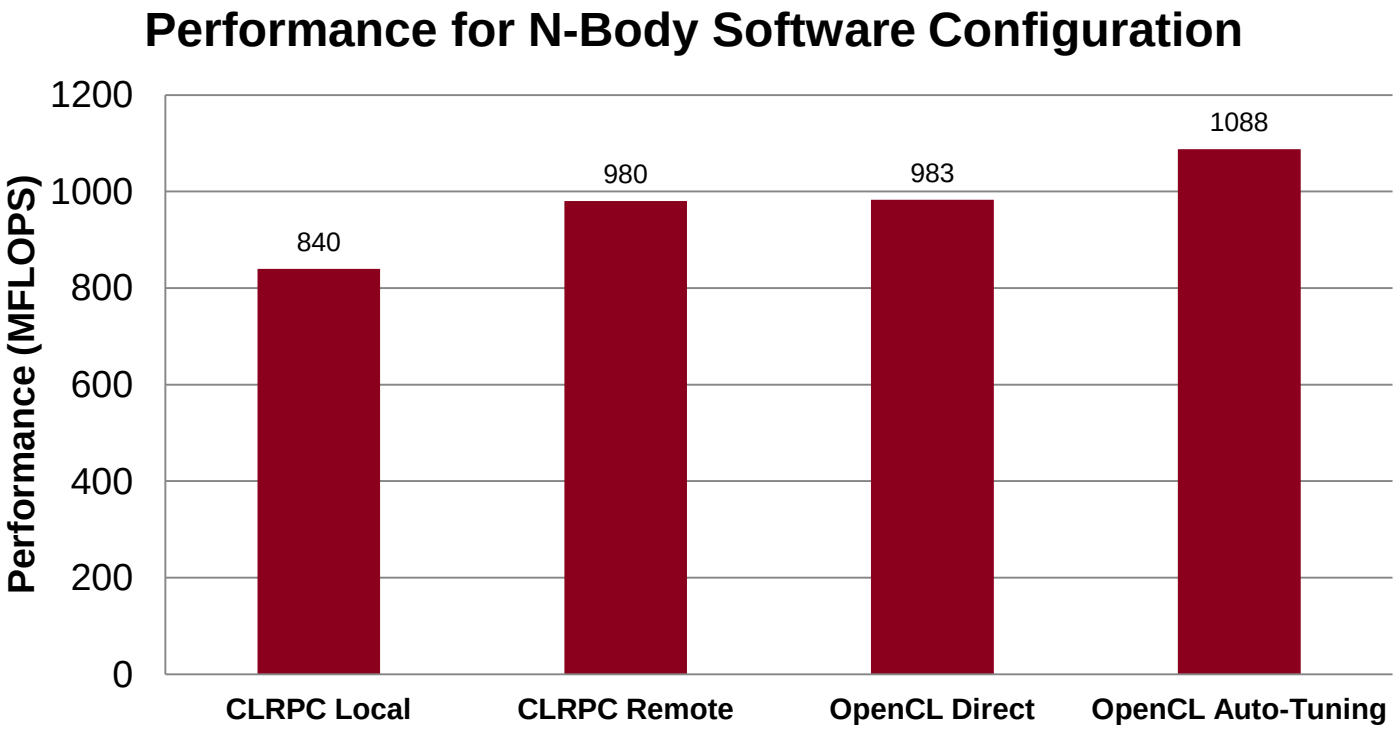
N-Body Benchmark

- We chose a gravitational n-body benchmark to flex the architecture
  - $O(N^2)$ compute and O(N) communication
  - Particle positions generate a force on all other particles
  - Accumulated acceleration changes particle velocity at each step
- Algorithm has two main steps:
  - Calculate total force on each particle
  - Update particle position/velocity over some small time-step (Newtonian dynamics)
  - Contains multiply, add, subtract, divide, and square root operations
- Entire (unoptimized) algorithm can be written in C with a few dozen lines of code

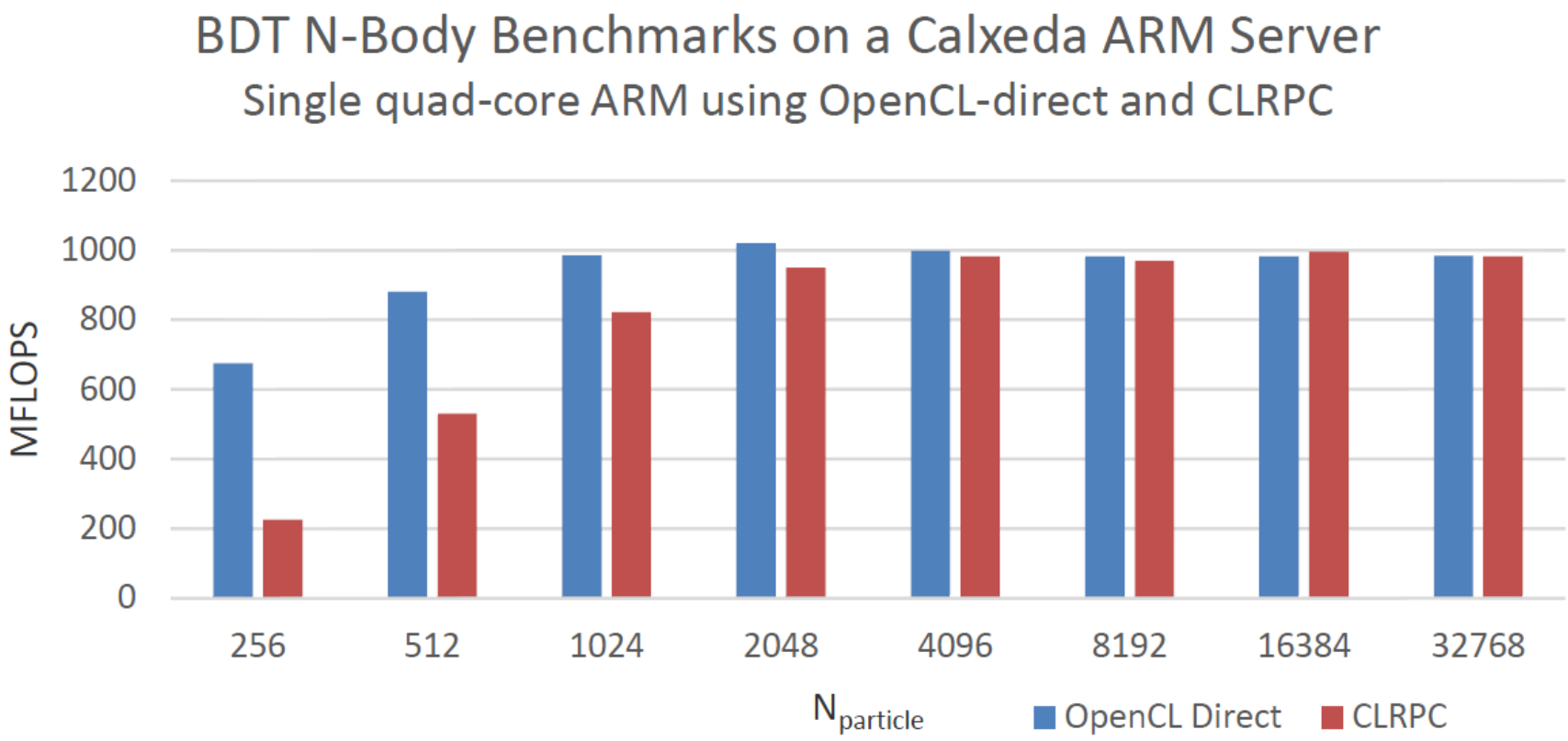


$$\mathbf{f}_i = \sum_{i \neq j} m_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3}$$

Benchmarks

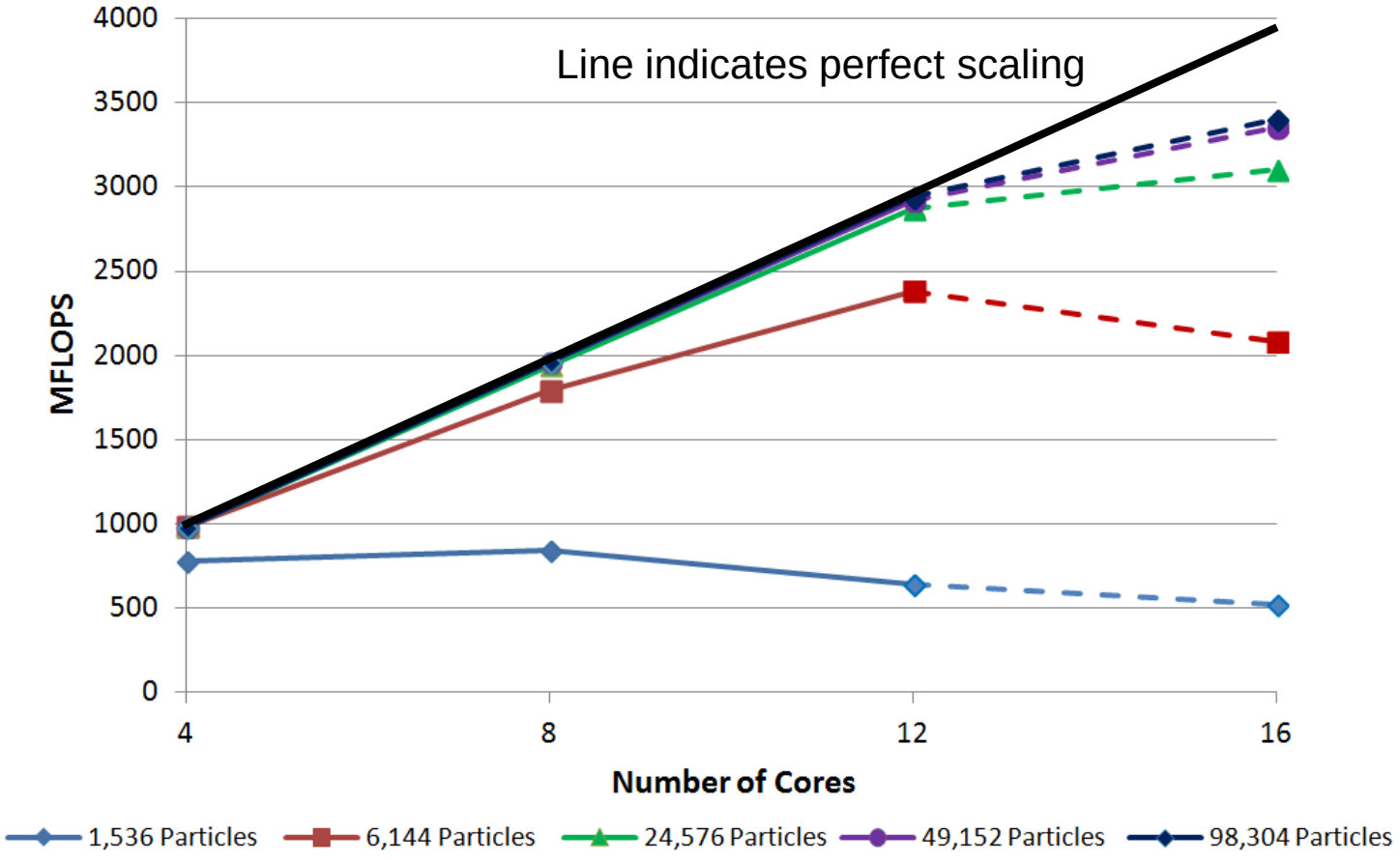

Performance for N-Body Software Configuration
Performance (MFLOPS)
1200
1000
800
600
400
200
0
840
980
983
1088
CLRPC Local
CLRPC Remote
OpenCL Direct
OpenCL Auto-Tuning

- A single node tells a story



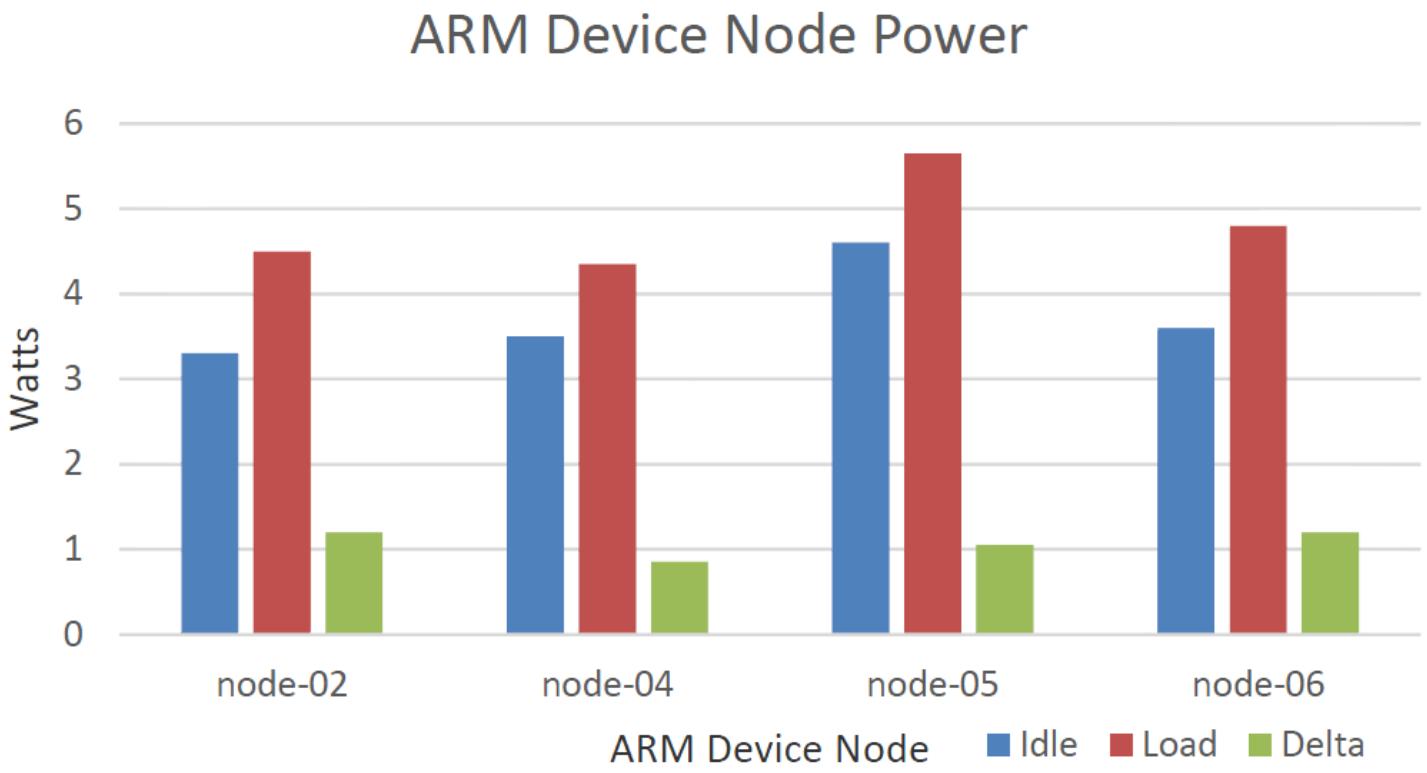
Benchmarks

- BDT N-Body Benchmarks on a Calxeda ARM Server
  - Scaling over 4 Quad-Core ARM SoCs Using a Single STDCL Context
  - Host is used for compute at 16 cores. This leads to performance issues



Power

- Idle power is a significant portion of load power (~80%)
- This is almost the opposite case for x86 servers where the CPU under load consumes a significant portion of the power budget



Conclusion and Future Work

- First effort to investigate OpenCL on Calxeda ARM server
- Novel OpenCL-based parallel programming model with abstraction to access networked compute devices
- Empirical results of n-body including auto-tuning code
- Power measurements during benchmarking insufficient to determine power efficiency rank compared to x86-based machines
- Future work includes investigating scaling on a much larger number of cores

References and Acknowledgments

- COPRTHR SDK:
  - https://github.com/browndeer/coprthr.git
- Exxact Corporation Calxeda Platform:
  - http://exxactcorp.com/index.php/solution/solu_list/59
- Khronos OpenCL:
  - http://www.khronos.org/opencl/

We wish to thank Exxact Corporation for providing access to their Calxeda based ARM server platform

EXXACT CORPORATION

www.exxactcorp.com