LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-648439

Performance Characteristics of HYDRA - a Multi-Physics simulation code from Lawrence Livermore National Laboratory

S. H. Langer, I. Karlin, M. M. Marinak

January 14, 2014

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Performance Characteristics of HYDRA - a Multi-Physics simulation code from LLNL

Steven H. Langer†, Ian Karlin†, Marty Marinak†

†Lawrence Livermore National Laboratory, Livermore, California 94551 USA
†{langer1, karlin1, marinak1}@llnl.gov

ABSTRACT

HYDRA is used to simulate a variety of experiments carried out at the National Ignition Facility (NIF) [4] and other high energy density physics facilities. HYDRA has packages to simulate radiation transfer, atomic physics, hydrodynamics, laser propagation, and a number of other physics effects. HYDRA has over one million lines of code and includes both MPI and thread-level (OpenMP and pthreads) parallelism.

This paper measures the performance characteristics of HYDRA using hardware counters on an IBM BlueGene/Q system. We report key ratios such as bytes/instruction and memory bandwidth for several different physics packages. The total number of bytes read and written per time step is also reported. We show that none of the packages which use significant time are memory bandwidth limited on a Blue Gene/Q. HYDRA currently issues very few SIMD instructions. The pressure on memory bandwidth will increase if high levels of SIMD instructions can be achieved.

Keywords

MPI, OpenMP, multi-physics, floating point performance

1. INTRODUCTION

Some physics simulation applications have a single physics module that consumes 95% or more of the run time. It is fairly easy to port a code like this to a new system and tune it so that it achieves good performance.

Other physics simulation applications have many physics packages. They often have multiple options for a given type of physics (e.g. radiation transport). These codes are used to run a wide range of simulations and only a subset of the physics packages are used in any given run. These codes are run in production mode by users who are not developers of the code. A code may have a hundred users at any given time.

Multi-physics applications may have over a million lines of code, are written by teams of 5-20 developers, use many external physics libraries, and have a lifetime that may exceed 20 years. The difficulties of writing and maintaining such a code require a different approach than a single-physics code. In particular, tuning for a specific system is not practical. Optimization efforts instead focus on characteristics of a whole class of systems.

These codes typically solve a set of coupled partial differential equations (PDEs) for time-dependent fields on a grid in three spatial dimensions. Variables like the temperature, density, and velocity depend only on the spatial coordinates. A large number of zones are often required due to the need to resolve small features.

The radiation field also depends on the photon energy, and 100-200 energy bins are often used. In terms of memory usage, the radiation field counts as 100-200 fields when radiation transport is treated using a diffusion approximation.

In other cases, detailed angular dependence of the radiation field is required. This is usually done using an S_N (discrete ordinates) method (see http://prod.sandia.gov/techlib/access-control.cgi/2002/021778.pdf and references cited therein). The radiation intensity is evaluated on a set of discrete directions in an S_N method. Resolving the angular dependence may require 1000 directions. Simulations with 100 thousand unknowns per zone are necessary for some key problems, so large amounts of memory are required to hold the state of the simulation.

The equations are often solved using the method of operator splitting. This essentially means having one function call (or one loop nest) for each term in the PDEs. There is a synchronization between all MPI processes at the end of each operator. This approach is referred to as "bulk synchronous" programming (see http://en.wikipedia.org/wiki/Bulk_synchronous_parallel and references cited therein). This approach produces a lower "high water mark" for memory usage than would occur if all operators were evaluated simultaneously.

Bulk synchronous programs have loops which are much simpler than if all terms were evaluated in a single very large loop. This makes the code easier to write and maintain (it is easier to see the relationship between the code and the mathematical operator). It also makes it easier for multi-person teams to work on a code. Each team member typically specializes in one or two areas of physics and does not need to look at code related to other physics packages. When a code contains over a million lines, it is important for code development to be efficient.

Lawrence Livermore National Laboratory (hereafter LLNL) has several multi-physics codes. They all have MPI parallelism and a subset of the physics packages also have thread-level parallelism. Running multi-physics codes effectively is the main requirement for a new LLNL parallel computer.

The operators in LLNL multi-physics codes are applied to full domains and the arrays they operate on are large compared to cache. That means each operator pulls its input arrays from DRAM into the cache. It then performs calculations, and stores the updated arrays back to DRAM.

Most fields are used by multiple operators, so they may make multiple round trips between DRAM and cache every time step. Operators using iterative methods may pull arrays into cache multiple times by themself. A system needs to have enough memory bandwidth to fetch arrays in a time short compared to the compute time for a single operator, not the compute time for a whole time step.

This paper examines one such multi-physics code, HYDRA, and documents its relative demands for floating point instructions, integer instructions, and memory bandwidth. This paper does not consider interconnect bandwidth or latency, although those characteristics are very important for some simulations.

2. HYDRA CHARACTERIZATION

HYDRA [3] [2] is a multi-physics code that simulates a variety of experiments conducted at the National Ignition Facility (NIF) and other pulsed laser facilities. The laser deposits a large amount of energy in a small volume, so HYDRA is focused on simulating the processes of high energy density physics. HYDRA has packages to simulate radiation transfer, atomic physics, hydrodynamics, laser propagation, and a number of other physics effects. HYDRA has over one million lines of code and includes both MPI and thread-level (OpenMP and pthreads) parallelism. Figure 2 shows the physics packages in HYDRA and their interconnections.

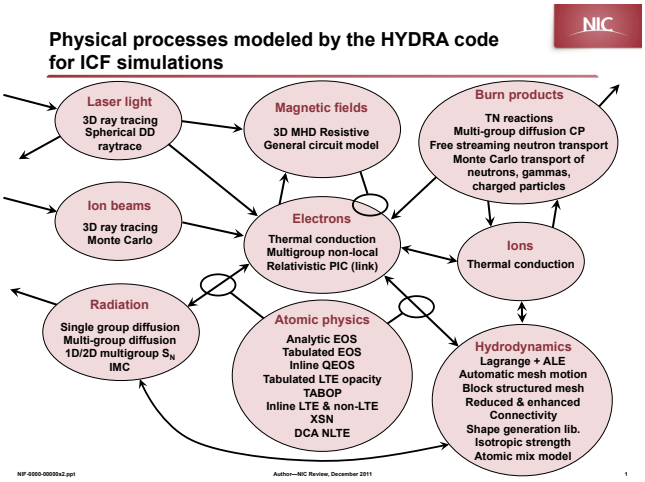

Figure 2: HYDRA has many physics packages so that it can simulate a broad range of experiments performed using the National Ignition Facility Laser.

HYDRA performs simulations on grids made up of one or more user blocks. User blocks usually correspond to major components of the object being simulated. For example, the capsule might be the first user block and the hohlraum wall the second user block in a NIF simulation. User blocks have curvilinear coordinates and the same topology as a regular 3D grid. This is sometimes referred to as an "ijk grid". There must be a one-to-one match of faces on adjacent user blocks. Within a user block, each zone is surrounded by exactly 26 other zones. This is also true for zones on a boundary face between two user blocks. The number of neighboring zones may be greater than or less than 26 for zones at the edge of a user block ("enhanced" and "reduced" connectivity). 2D simulations are run by limiting the width in one direction to a single zone.

HYDRA uses domain decomposition of the spatial grid to implement MPI parallelism. User blocks are decomposed into multiple ijk domains. All zones in a domain may be accessed using three indices in HYDRA. Some other multi-physics codes at LLNL use arbitrarily connected grids and their zones are accessed by iterating through lists of zones rather than indexing a 3D array.

All major physics packages also have thread-level parallelism. In the case of hydrodynamics and some other packages, threading is over domains. If there are 4 hardware threads available per MPI process, the user requests 4 domains per process and one thread handles each domain. This threading is implemented via OpenMP directives and is done at such a high level that OpenMP thread synchronization time is not an issue.

There are a number of important physics packages where the computational cost of updating a zone varies by large amounts across the full grid. This leads to a load imbalance between the domains. A more complex threading approach is required to deal with load imbalance.

The DCA package computes frequency-dependent opacities for all zones in the grid. The work in some zones is much greater than in others, particularly when the matter is not in local thermodynamic equilibrium. Putting simple OpenMP directives on loops would not alleviate the load imbalance. The DCA package therefore varies the number of OpenMP threads per domain (based on timing information from the last time step) so that the work per thread is roughly constant. As an example, HYDRA might have 8 MPI domains on an Intel Sandy Bridge node with 16 cores and 32 hardware threads. If one domain has much more DCA work than the other seven, it might be assigned 32 OpenMP threads while the other domains have one OpenMP thread each. This approach evens out the work per hardware thread on a single node, but it does not help when there is a large imbalance in the DCA work on different nodes. It is not possible to dynamically move hardware threads from one process to another on a BGQ system, so the load balancing in DCA is turned off.

The laser and IMC (Implicit Monte Carlo) packages in HYDRA use thread parallelism implemented using pthreads. Each active process has a thread which handles all MPI message passing, another thread handles all updates of the energy deposition array (recording the net transfer of energy between the matter in a zone and the laser rays or IMC particles passing through it). If a process has a large number of threads, it is possible to have more than one message passing or deposition thread.

Each active process is assigned several domains either by idling a portion of the MPI processes or by having multiple domains per process. The time spent processing each domain is recorded on every time step. At the start of the next time step, a genetic algorithm shuffles domains around until all active processes have roughly the same amount of work when summed over all their domains. For example, a process assigned a "difficult" domain will also be assigned several "easy" domains. In practice, load balancing works well with 4 or more domains per active process.

3. STUDIES ON BLUE GENE/Q

IBM's Blue Gene/Q was chosen as the system on which

to gather performance data. The HPM library written by Bob Walkup of IBM provides a simple way to gather the desired hardware counters. The hardware counters on a BGQ system are easy to understand due, in part, to the simple in-order cores.

The main time step function of HYDRA was modified to have HPM_Start() and HPM_Stop() calls around each physics package. When the job terminates, HPM writes output files containing key performance metrics for each node. These include the number of cycles, integer instructions, floating point instructions, and floating point operations for each package. HPM also reports the number of L2 misses for each node. An L2 miss triggers a load of a 128 byte cache line on a read miss, so L2 cache misses may easily be turned into the number of bytes fetched from DRAM. HPM also reports the number of cache lines flushed to DRAM, which allows the number of bytes written to DRAM to be computed.

Table 1 reports performance metrics from BGQ runs of several applications. HYDRA has already been described. The pF3D kernels were extracted from pF3D, a code which simulates laser-plasma interactions in NIF experiments. pF3D simulations are often run with more than 100,000 cores. pF3D has fewer packages than HYDRA, but has many more performance critical loops than a single physics code. MCB is a Monte Carlo radiation transport mini-app used in evaluating new computer systems. microK is a set of microkernels which operate on vectors. microK is helpful in evaluating cache behavior and compiler optimizations. microK results are reported for vectors which fit in on chip cache memory and for vectors large enough that they must be fetched from DRAM.

A BGQ chip has 16 cores available to the user. Each core has 4 hardware threads. It requires at least two threads per core to reach the maximum instruction issue rate. The maximum number of integer instructions per node per cycle is 16, as is the maximum number of floating point instructions. The clock speed is 1.6 GHz and the chip uses the PowerPC instruction set. The BGQ has a 4-wide SIMD floating point unit and has a fused multiply-add (FMA) instruction. Floating point instructions may perform from 1 to 8 floating point operations.

The integer unit on the BGQ chip handles loads, stores, integer arithmetic, address computations, and a number of other instructions. Codes operating on arrays of floating point numbers will issue a lot of integer instructions as they load and store array elements and compute addresses.

The numbers in the tables are derived from hardware counters on the BGQ chip. The "packages" in the table have from 1.3 to 8 floating point operations per instruction, so they span nearly the full possible range. The BGQ compiler does a good job of generating FMA instructions. The BGQ compiler has difficulty generating SIMD instructions unless the code is annotated with BGQ-specific alignment directives. The loops in microK are so simple that we added directives and achieve a high SIMD fraction. The dot product kernel, for example, issue mostly 4-wide FMA instructions. It is impractical to add those directives to a large code, so the SIMD fraction for HYDRA and pF3D is essentially zero. Some of the pF3D kernels deliver more than 2 FLOPs per instruction because they call IBM's "hand written" sin, exp, etc. special functions. The compiler turns the "divide two vectors" kernel into a call to a vector divide function. That function uses reciprocal approximation followed by Newton-Raphson iteration. These instructions can all be pipelined whereas the hardware divide instruction cannot. The result is that the divide kernel has fairly high computational intensity.

The microK run used one MPI process with 32 OpenMP threads on a single node. The short vector case had 64K elements per thread and the large vector case had 512K elements per thread.

The microK results demonstrate the performance impact of memory bandwidth. The short vectors run entirely out of cache and the polynomial kernel achieves over 50% of the peak floating point performance of the node. The large vector run fetches its operands from memory and is memory bandwidth limited in all cases (the bandwidth is 23 to 27 GB/s compared to the streams bandwidth of 28 GB/s).

The polynomial evaluation and divide vector micro-kernels execute more floating point instructions than integer instructions. HYDRA, MCB, the rest of microK, and the pF3D kernels all execute more instructions in the integer unit than in the floating point. MCB performs a lot of its computations using integer arithmetic and has a very low fraction of floating point instructions. HYDRA's laser and multi-group diffusion package have an even lower floating point fraction than MCB.

All HYDRA tests were run with 64 processes running on 16 BGQ nodes (4 per node). The 64 hardware threads on a node were equally divided amongst the 4 processes.

The hyd607 test problem performs a capsule-only simulation of a NIF implosion experiment. Most of the time is spent in the multi-group diffusion package (mtgrdif), with roughly 10% of the time spent in electron heat conduction, advection, and evaluation of the equation of state and opacities. mtgrdif has a high fraction of integer instructions. The 4 processes running on a node used a total of 1.7 GB of heap memory. The radiation diffusion package transfers 14.7 GB between DRAM and the processor during a time step, so some arrays are read multiple times. The diffusion package solves a large sparse matrix using an iterative scheme, so arrays are naturally fetched multiple times if they are too big to fit in cache.

The nifburn test problem performs a simulation of the capsule and the surrounding hohlraum for a NIF experiment. Domain replication was employed so that HYDRA could load balance the laser and imc packages across processes. Most of the time is spent in the laser ray trace and Implicit Monte Carlo radiation transport packages. The hydrodynamics package, advection associated with ALE remaps, electron heat conduction, and fusion burn combine to consume roughly 15% of the run time.

The 4 processes running on a node in nifburn used a total of 2.7 GB of heap memory. The imc package transfers 12.97 GB between DRAM and the processor during a time step, so some arrays are read multiple times. The opacity array has nzones times ngroups elements and is much larger than the cache.As the Monte Carlo particles randomly wander through the grid, they will pull the opacity array in multiple times.

The tables include ratios of cache misses to lines read. A cache line is 128 bytes on a BGQ system. HYDRA performs most computations using double precision operands, so a line holds 16 numbers. A stride one loop should have one cache miss per L2 cache line read (the BGQ counts a cache

Package	int inst per cycle	FPinst per cycle	FLOP per cycle	DRAM BW (GB/s)	DRAM xfer GB	Bytes per inst	L2 miss per line
HYDRA hyd607							
advect	5.74	1.90	1.59	3.21	2.13	0.230	1.31
eosOpac	1.70	0.48	1.32	0.46	0.146	0.124	2.17
econd	10.03	0.39	1.53	0.94	0.85	0.056	0.86
mtgrdif	8.74	0.25	1.58	0.85	14.65	0.058	0.73
HYDRA nifburn							
hydro	4.38	2.21	1.62	1.89	0.31	0.148	1.58
advect	4.68	0.58	1.62	2.91	0.72	0.32	1.33
econd	12.66	0.95	1.66	0.32	0.10	0.014	2.89
laser	1.69	0.05	1.25	3.02	6.93	1.08	4.31
imc	1.06	0.21	1.60	1.29	12.93	0.577	5.27
burn	12.40	0.72	1.60	0.17	0.29	0.008	1.03
MCB							
advance	4.52	0.18	1.31	0.25	4.92	0.033	1.01
pf3d kernels							
couple4	4.02	1.47	2.81	1.46	12.32	0.112	0.45
absorbdt	4.61	1.13	1.77	1.21	1.23	0.114	0.43
acadv	3.56	1.15	2.256	1.87	9.67	0.191	0.28
advancefi	5.29	1.78	2.12	0.88	3.07	0.061	0.48
fft	3.02	1.88	1.30	2.64	1.79	0.303	0.48
microK small vector							
sdot	8.16	1.90	8.00	1.49	0.03	0.040	0.38
poly	4.04	9.28	8.00	0.43	0.008	0.003	12.60
divide	5.90	8.94	5.60	0.0003	6e-6	3.5e-6	0.78
microK large vector							
sdot	8.16	1.90	8.00	1.49	0.03	0.040	0.38
poly	4.04	9.28	8.00	0.43	0.008	0.003	12.60
divide	5.90	8.94	5.60	0.0003	6e-6	3.5e-6	0.78

Table 1: This table shows performance metrics for two HYDRA test problems. Metrics from three other codes are shown for reference. HYDRA simulations used 64 MPI processes on 16 BGQ nodes (4 per node). All HYDRA, pF3D, and MCB packages issue many more integer instructions than floating point instructions. The ctr-chk polynomial and divide vector micro-kernels issue more floating point than integer instructions. HYDRA, pF3D, and MCB use well below the node bandwidth of 28 GB/s. The micro-kernels are all memory bandwidth limited for large vectors.

miss whether or not a prefetch occurred). A package which accesses large arrays randomly might have up to 16 misses per line. HYDRA's Implicit Monte Carlo (imc) package has a higher miss fraction than any other package in the table. That is not surprising given that the particle list has photons almost randomly scattered through the grid at the time the performance counters were read.

All HYDRA packages run well under the peak floating point performance. The performance bottleneck has not yet been identified. The L2 cache latency is much larger on a BGQ than for x86_64 systems, so that might be a problem.

3.1 Memory Usage

The hyd607 test problem uses a total of 1.7 GB of heap memory per node. The radiation diffusion package transfers 14.7 GB between DRAM and the processor during a time step, so some arrays are read multiple times. The diffusion package solves a large sparse matrix using an iterative scheme, so arrays are naturally fetched multiple times if they are too big to fit in cache.

The nifburn test problem uses a total of 2.7 GB of heap memory per node. The imc package transfers 12.97 GB between DRAM and the processor during a time step, so some arrays are read multiple times. The opacity array has nzones times ngroups elements and is much larger than the cache.As the Monte Carlo particles randomly wander through the grid, they will pull the opacity array in multiple times.

The data we have gathered does not establish whether hyd607 and nifburn suffer a performance penalty due to pulling arrays in from DRAM multiple times. There is a lot of unused DRAM bandwidth, but DRAM latency might be an issue. This is particularly true for the IMC where the prefetch hardware won't be able to figure out which zone to fetch next. Adding SIMD instructions would increase the pressure on memory bandwidth, but both nifburn and hyd607 are running at well under one quarter of the memory bandwidth.

Future systems will have a lower ratio of DRAM bandwidth to peak floating point performance. If the ratio drops too much relative to current systems, DRAM bandwidth could become a performance constraint. Future systems may have in package memory (IPM) with a bandwidth significantly higher than off chip DRAM bandwidth. hyd607 and nifburn could both see a benefit from using IPM as a cache because they currently pull arrays in from DRAM multiple times per call.

IPM will probably have latencies only slightly less than off chip DRAM. It will be hard to predict the possible benefits of IPM without understanding the importance of latency. That will be a key topic in our future performance analysis work.

4. CONCLUSION

Our goal in this work was to investigate the performance

characteristics of HYDRA, a multi-physics simulation code from LLNL. Integer and floating point operation counts, bytes read and written from DRAM, and memory bandwidths were reported for several physics packages. We demonstrated that, due to the operator splitting approach, the total memory traffic per time step between the processor chip and DRAM is significantly greater than the total amount of memory in use by HYDRA. All current multi-physics codes at LLNL have this characteristic.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work was funded by the Laboratory Directed Research and Development Program at LLNL under project tracking codes 13-ERD-055 and 13-FS-002 (LLNL-CONF-635776).

The authors would like to thank Mehul Patel and Scott Sepke of LLNL for assistance in running our HYDRA test problems and for explanations of how HYDRA physics packages are implemented.

5. REFERENCES

[1] T. Jones, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Caffrey, B. Maskell, P. Tomlinson, and M. Roberts. Improving the Scalability of Parallel Jobs by Adding Parallel Awareness to the Operating System. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing (SC'03)*, 2003.

[2] M. Marinak, G. Kerbel, J. Koning, M. Patel, S. Sepke, M. McKinley, M. OÕBrien, R. Procassini, and D. Munro. Advances in hydra and its applications to simulations of inertial confinement fusion targets. In *Proceedings of the 2011 International Fusion Sciences and Applications Conference (IFSA 2013)*, volume 59 of *IFSA*, page 3011. EPJ Web of Conferences, 2013.

[3] M. M. Marinak, G. D. Kerbel, N. A. Gentile, O. Jones, D. Munro, S. Pollaine, T. R. Dittrich, and S. W. Haan. Three-dimensional hydra simulations of national ignition facility targets. *Physics of Plasmas*, 8(4):22755, Apr. 2001.

[4] E. I. Moses, R. N. Boyd, B. A. Remington, C. J. Keane, and R. Al-Ayat. The national ignition facility: Ushering in a new age for high energy density science. *Phys. Plasmas*, 16(041006):1–13, April 2009.