



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Performance Characteristics of HYDRA - a Multi-Physics Simulation Code from LLNL

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

January 12, 2015

VECPAR 2014
Eugene, OR, United States
June 30, 2014 through July 3, 2014

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Performance Characteristics of HYDRA - a Multi-Physics Simulation Code from LLNL

Steven H. Langer, Ian Karlin, and Michael M. Marinak

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

Abstract. HYDRA simulates a variety of experiments carried out at the National Ignition Facility and other high energy density physics facilities. It 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, includes MPI and thread-level (OpenMP and pthreads) parallelism, has run on a variety of platforms for two decades, and is undergoing active development.

In this paper, we demonstrate that HYDRA’s thread-based load balancing approach is very effective. Hardware counters from IBM Blue Gene/Q runs show that none of HYDRA’s packages are memory bandwidth limited, a few come close to the maximum integer instruction issue rate, and all are well below the maximum floating point issue rate.

Topics: Large-scale Simulations in CS&E, Multiscale and Multiphysics Problems, Performance Analysis.

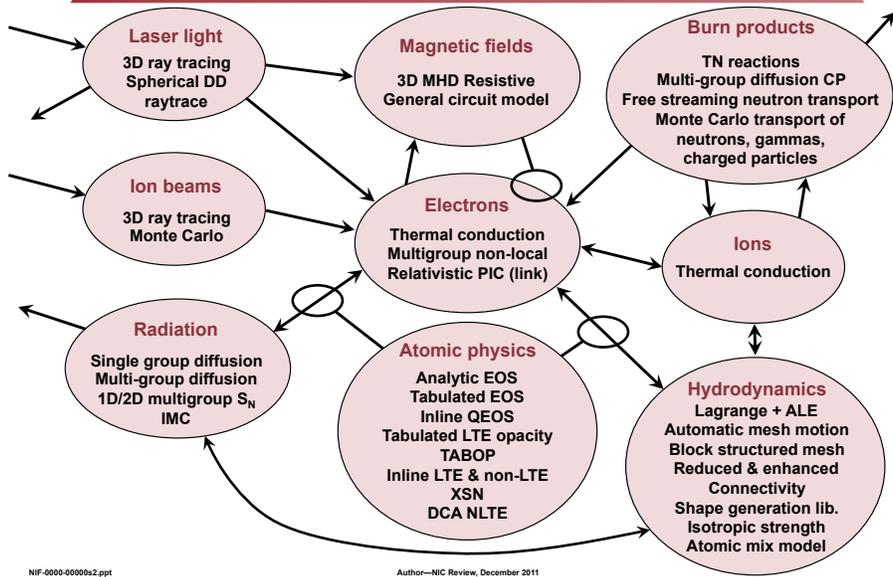
1 HYDRA - A Multi-Physics Simulation Code

The goal of this paper is to introduce readers to a complex “multi-physics” code, discuss some of the techniques used to improve performance, and use data from hardware counters to provide insight into the bottlenecks controlling the performance. We chose HYDRA [4] [5], which is used to simulate experiments conducted at the National Ignition Facility (NIF) [6] and other pulsed laser facilities, as our test code. 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 is a “multi-physics” simulation code. Figure 1 shows the many physics packages in HYDRA and their interconnections. HYDRA has characteristics similar to other multi-physics codes at LLNL. It consists of over a million lines of code, has run on a variety of platforms for two decades, and is still undergoing active development. HYDRA runs a wide range of simulations and only a subset of the physics packages are used in any given run.

This is the first paper to present a performance analysis of HYDRA. Single-physics codes may have a single loop which consumes over 90% of the run time.

Physical processes modeled by the HYDRA code for ICF simulations



NIF-0000-00000s2.ppt Author—NIC Review, December 2011 1

Fig. 1. HYDRA has many physics packages so that it can simulate a broad range of experiments, including those performed on the National Ignition Facility Laser.

HYDRA and other multi-physics codes have dozens of “hot loops”, and the hot loops change from one run to the next. Multi-physics codes use a programming approach focused on portability and programmer productivity. Performance is very important, but optimizations need to work for a variety of systems. Modifying the code to increase the percentage of stride one accesses or enable generic SIMD utilization is worthwhile, but tuning for a particular SIMD unit is not.

HYDRA solves a set of coupled partial differential equations (PDEs) for time-dependent fields on a grid in three spatial dimensions. A large number of zones are often required to resolve small features. The temperature, density, and velocity depend only on the spatial coordinates, but the radiation field also depends on the photon energy. Simulations often use 100-200 energy bins so the radiation field and opacity arrays may dominate memory usage. The equations are solved using the method of operator splitting [2]. This essentially means having one function call (or one loop nest) for each term in the PDEs.

There is synchronization between all MPI processes at the end of each operator. This approach is referred to as “bulk synchronous” programming [8]. 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. Each team member specializes in a few areas of physics and rarely needs to look at code related to other physics packages.

The operators in HYDRA 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. 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.

2 HYDRA characterization

HYDRA uses a block structured mesh. The mesh has one or more user blocks which 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 a regular 3D grid topology. This is sometimes referred to as an “ijk grid”. There is a one-to-one match of faces on adjacent user blocks. Most zones are surrounded by exactly 26 other zones. The number of neighboring zones may be more or less than 26 at the corner of a user block (“enhanced” and “reduced” connectivity). HYDRA uses domain decomposition of the spatial grid to implement MPI parallelism. User blocks are decomposed into multiple ijk MPI domains.

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 a level high enough 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 from domain to domain. The regions with the highest work load shift throughout the course of a run. Some HYDRA packages use a more complex threading approach to deal with this type of load imbalance.

The DCA package computes frequency-dependent opacities for all zones. Some zones require more work than others, particularly when the matter is not in local thermodynamic equilibrium. The DCA package varies the number of OpenMP threads per domain (based on timing 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 threads while the other domains have one 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. Threads are statically bound to processes on a BGQ system, so the dynamic load balancing in DCA is turned off.

The threads in HYDRA’s laser ray trace and IMC (Implicit Monte Carlo) packages cooperatively process a set of domains. On Linux clusters, two MPI processes per node are typically designated as ”masters”. The other processes on a node send their domains to the master processes and then become inactive. On Blue Gene/Q systems, each process makes several “replicates” of its domain. The IMC particles or laser rays for a domain are split among the replicates. In either case, the active processes have several domains. A genetic algorithm shuffles domains around until each process has a nearly constant amount of work (based on the last time step). A process assigned a “difficult” domain is also assigned several “easy” domains. Load balancing works well with 4 or more domains per active process.

These packages use pthreads with thread specialization. Each active process has a thread which handles all MPI message passing, so a thread safe MPI is not required. 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), so locks on the deposition array are not required. The remaining threads push IMC photons or trace laser rays.

3 Dynamic Load Balancing

The physics equations solved by HYDRA often require more work in some zones than in others. If the problem is divided up into equal sized domains and no provision is made for load imbalance, the run time will be much longer than for a well balanced job. The dynamic load balancing results presented below were obtained from a standard NIF hohlraum simulation on a cluster with dual socket Intel Sandy Bridge nodes and an Infiniband QDR interconnect.

Table 1. This table shows the time without and with dynamic load balancing for the laser, IMC, and DCA packages during a HYDRA NIF simulation. The speedup for the laser package was greater earlier in the run. The final line is the total physics time for the run. A 50% speedup in the physics time is much appreciated by HYDRA’s users.

Package	Time	Time	Time speedup ratio
	(sec)	(sec)	
	no balance	balanced	
laser	162.2	130.6	1.24
IMC	392.6	234.4	1.68
DCA	6.8	3.6	1.90
total	557.9	363.4	1.51

The results in Table 1 show that load balancing cuts the run time for some packages almost in half and reduces the overall run time of the NIF simulation to roughly two thirds of what it would be without load balancing. This is a large improvement compared to what is typically obtained by adjusting compiler optimization flags. The load balance varies from one simulation to the next. In one recent simulation, DCA load balancing reduced the run time to roughly 10%

of the time with no load balancing (i.e. a 10X speedup). Other multi-physics codes might benefit from adopting a load balancing approach like HYDRA's.

4 Studies on Blue Gene/Q

4.1 BGQ Overview

IBM's Blue Gene/Q was chosen as the system on which to gather performance data. The clock speed is 1.6 GHz and the chip uses the Power instruction set. A BGQ chip has 16 cores with 4 hardware threads each. Two threads per core are required to reach the maximum instruction issue rate of 16 integer instructions and 16 floating point instructions per node per cycle. 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 L1 cache is 16 kB per core. Each core has a 2 MB slice of L2 cache operated as part of a 32 MB shared cache. It has a high latency (roughly 50 ns) because it uses eDRAM and requires extra logic to bind the slices together. The Intel Sandy Bridge has a 11 ns latency to its L3 cache. 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 issue many integer instructions as they load and store array elements and compute addresses. A BGQ system has a streams bandwidth of 28 GB/s per node. The memory space is flat, so there is no need to worry about NUMA effects.

4.2 The BGQ Test Codes

HYDRA tests were run with 4 processes per node on 16 BGQ nodes. The 64 hardware threads on a node were equally divided amongst the 4 processes. Data from three other codes is provided to help assess whether HYDRA has unique performance bottlenecks.

pF3D[7, 1, 3] is a massively parallel code which simulates laser-plasma interactions in experiments using the National Ignition Facility laser and other high power lasers. pF3D has fewer packages than HYDRA, but still has 2 dozen performance critical loops. pF3D operates on 3D arrays which makes it easy to use stride one memory access in many (but not all) loops.

MCB is a Monte Carlo mini-app used in investigating new computer systems and new programming approaches. It is dominated by integer computation and has the erratic branching character of all Monte Carlo radiation transport codes.

microK is a set of simple vector loops used to measure the performance impact of falling out of cache, speedups due to using SIMD instructions, and other processor features. The loops in microK are simple so it is fairly easy to optimize them on a new system. The microK runs used one MPI process with 32 OpenMP threads on a single node.

Table 2. This table shows performance metrics for two HYDRA test problems. Metrics from three other codes are shown for reference. The polynomial kernel is the only one which issues more floating point than integer instructions. The polynomial and dot product kernels use nearly the full memory bandwidth (28 GB/s) for large vectors, but all other tests use no more than 11%. For small vectors, microK runs completely out of the L2 cache so we do not report any DRAM related numbers.

Package	Time (sec)	int Instr per cycle	FP Instr per cycle	FLOP per Instr	DRAM BW (GB/s)	DRAM xfer (GB)	L2 miss per line
hydra hyd607							
advect	0.66	5.74	1.90	1.59	3.21	2.13	1.31
eosOpac	0.32	1.70	0.48	1.32	0.46	0.146	2.17
econd	0.90	10.03	0.39	1.53	0.94	0.85	0.86
mtgrdif	17.20	8.74	0.25	1.58	0.85	14.65	0.73
hydra nifburn							
hydro	0.17	4.38	2.21	1.62	1.89	0.31	1.58
advect	0.29	4.68	0.58	1.62	2.91	0.72	1.33
econd	0.32	12.66	0.95	1.66	0.32	0.10	2.89
laser	2.29	1.69	0.05	1.25	3.02	6.93	4.31
imc	10.06	1.06	0.21	1.60	1.29	12.93	5.27
burn	1.65	12.40	0.72	1.60	0.17	0.29	1.03
MCB							
advance	19.52	4.52	0.18	1.31	0.25	4.92	1.01
pf3d kernels							
couple4	8.43	4.02	1.47	2.81	1.46	12.32	0.45
absorbdtd	1.02	4.61	1.13	1.77	1.21	1.23	0.43
acadv	5.16	3.56	1.15	2.25	1.87	9.67	0.28
advancefi	3.47	5.29	1.78	2.12	0.88	3.07	0.48
fft	0.68	3.02	1.88	1.30	2.64	1.79	0.48
microK small							
sdot	0.02	8.16	1.90	8.00			
poly	0.02	4.04	9.28	8.00			
microK large							
sdot	0.57	1.96	0.46	8.00	23.73	13.48	0.51
poly	0.51	1.09	2.56	8.00	26.21	13.42	1.42

4.3 Performance Metrics

The HPM library written by Bob Walkup of IBM provides a simple way to gather the desired hardware counters. HPM start and stop calls were added around calls to physics package in the time step loop. HPM reports how many times each event occurred in a package. An L2 miss is recorded for every 128 byte cache line loaded and a flush for every line stored. This makes it easy to calculate the DRAM read and write bandwidth.

Table 4.2 reports performance metrics from BGQ runs of the test applications. Two HYDRA test problems were run to show how the time spent in physics packages and the set of packages used varies from problem to problem.

The hyd607 test problem performs a capsule-only simulation of a NIF implosion experiment. Most time is spent in the multi-group diffusion package

(mtgrdif), with roughly 10% of the time spent on electron heat conduction, advection, equation of state, and opacities. The nifburn test problem performs an integrated simulation of the capsule and the surrounding hohlraum for a NIF experiment. Domain replication was employed to allow load balancing of the laser and IMC packages. Most time is spent in the laser and IMC packages. The hydrodynamics package, advection associated with ALE remaps, electron heat conduction, and fusion burn combine to consume about 15% of the run time.

The BGQ compiler generates a fairly high fraction (30% or more) of FMA instructions for all test codes. The BGQ compiler has difficulty generating SIMD instructions unless the code is annotated with BGQ-specific alignment directives. The simple loops in microK allowed us to add alignment directives and achieve nearly a 100% SIMD fraction. It is impractical to add those directives to a large code, so the SIMD fraction is low for HYDRA, pF3D, and MCB. Some of the pF3D kernels deliver more than 2 FLOPs per instruction because they call IBM's "hand written" sin, exp, etc. special functions.

Floating point instruction issue rates are not a bottleneck for HYDRA, pF3D, or MCB. Only the polynomial kernel issues more floating point instructions than integer instructions (on the BGQ). HYDRA's econd, burn and mtgrdif packages and the dot product of short vectors all execute more than 8 integer instructions per cycle and their performance may be limited by integer issue rates.

MicroK results are reported for vectors which fit in the L2 cache (64K elements per thread) and for vectors large enough (512K elements per thread) that they must be fetched from DRAM. The polynomial kernel achieves over 50% of the peak floating point performance for short vectors but only 16% of peak for long vectors. The microK kernels are memory bandwidth limited for large vectors (the bandwidth is close to the 28 GB/s streams bandwidth). The highest memory bandwidth for HYDRA, pF3D, or MCB is 3.2 GB/s, so memory bandwidth is not a bottleneck for "production" codes.

The high cache and DRAM latency on a BGQ hurts the performance of the IMC, laser ray trace, and EOS and opacity lookup packages in HYDRA. The 50 ns latency to the L2 cache on a BGQ is much larger than the 11 ns latency to the L3 cache on a Sandy Bridge. The L1P unit on a BGQ prefetches from L2 to L1 to try and hide latency. The L1P hit rate for the IMC and laser packages is 1-2%. HYDRA suffers a 50 ns delay on almost all L2 accesses, and the effective bandwidth of L2 will be low. The DRAM latency on a BGQ is 220 ns versus 70 ns for a Sandy Bridge. The high latency on the BGQ will prevent HYDRA from achieving full memory bandwidth unless prefetching from DRAM to L2 works well. We have not measured the DRAM prefetch efficiency. The laser rays and IMC particles move from zone-to-zone in a manner which is hard to predict, so we expect that prefetch efficiency will be low.

The pF3D kernels issue from 3 to 5.3 integer instructions per cycle. That is low enough that they probably are not bottlenecked on the integer issue rate. The pF3D kernels all issue at least one floating point instruction per cycle per node which is several times more than the laser and IMC packages in HYDRA, but far below the peak. The pF3D kernels have low L2 miss rates because many

inner loops access arrays in stride one order. It is not clear what is the key performance bottleneck for the pF3D kernels.

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. A package which accesses large arrays randomly might have up to 16 misses per line. HYDRA’s 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 scattered almost randomly through the grid at the time the performance counters were read.

4.4 Memory Usage

The hyd607 test problem uses 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, which shows that some arrays are read multiple times. The diffusion package solves a large sparse matrix using Hypr’s iterative CG solver with hybrid AMG/diagonal pre-conditioner. The iteration is the reason arrays are fetched multiple times.

The nifburn test problem uses 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. As the Monte Carlo particles randomly wander through the grid, they will pull the opacity array in multiple times.

HYDRA either has enough memory bandwidth on the BGQ or (more likely) is bottlenecked by memory latency. Future systems will have a lower ratio of DRAM bandwidth to peak performance. To deal with this “memory wall”, these systems may include some in-package memory (IPM). IPM bandwidth will be much higher than external DRAM bandwidth, but its latency will be similar. If memory latency is the key bottleneck for HYDRA, IPM may not help performance significantly.

5 Conclusion

Our goal in this work was to investigate the performance characteristics of HYDRA, a multi-physics simulation code. We expect that other multi-physics codes from LLNL will have similar characteristics. We demonstrated that HYDRA’s thread based load balancing strategy is very effective. DRAM bandwidth is not a limiting factor for any HYDRA package. The latency of DRAM or the L2 cache may be a bottleneck. Floating point issue rates are never a limiting factor for HYDRA, but integer issue rates may be a limiting factor for a few packages.

We demonstrated that the total memory traffic between the processor chip and DRAM is significantly greater than the total amount of memory in use by HYDRA, indicating that using IPM as a cache may have benefits for HYDRA on future systems.

Prepared by LLNL under Contract DE-AC52-07NA27344, LLNL-PRES-652559.

References

1. R. L. Berger, B. F. Lasinski, A. B. Langdon, T. B. Kaiser, B. B. Afeyan, B. I. Cohen, C. H. Still, and E. A. Williams. Influence of spatial and temporal laser beam smoothing on stimulated brillouin scattering in filamentary laser light. *Phys. Rev. Lett.*, 75(6):1078–1081, Aug 1995.
2. H. Holden, K. H. Karlsen, K.-A. Lie, and H. Risebro. *Splitting Methods for Partial Differential Equations with Rough Solutions*. European Mathematical Society, Zurich, Switzerland, 2010.
3. S. Langer, B. Still, T. Bremer, D. Hinkel, B. Langdon, and E. A. Williams. Cielo full-system simulations of multi-beam laser-plasma interaction in nif experiments. *CUG 2011 proceedings*, 2011.
4. M. Marinak, G. Kerbel, J. Koning, M. Patel, S. Sepke, M. McKinley, M. O'Brien, 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.
5. 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.
6. 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.
7. C. H. Still, R. L. Berger, A. B. Langdon, D. E. Hinkel, L. J. Suter, and E. A. Williams. Filamentation and forward brillouin scatter of entire smoothed and aberrated laser beams. *Physics of Plasmas*, 7(5):2023–2032, 2000.
8. L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, Aug. 1990.