Program Analysis Tools for Application-Specific Architectures

Maya B. Gokhale, maya@lanl.gov
Matthew J. Sottile, matt@lanl.gov

Los Alamos National Laboratory

Outline

- Scientific Simulations
- Heterogeneous clusters
- Application analysis
- Application characterization tools
- Conclusions

Supercomputing Applications

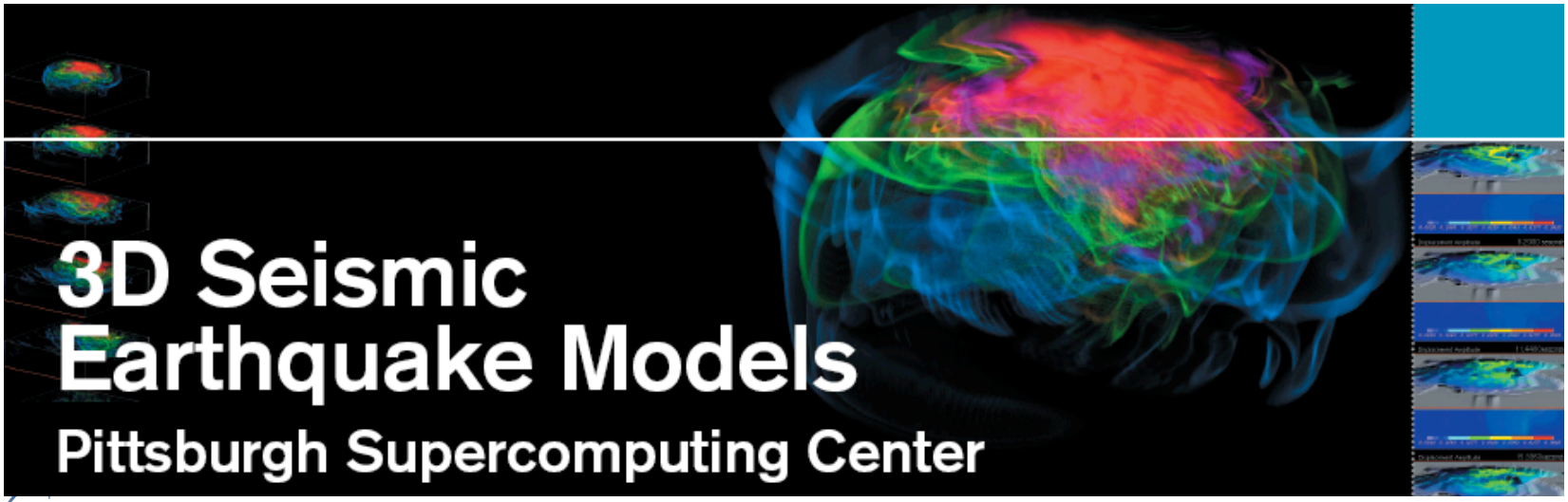
- Analysis
  - bioinformatics - BLAST, genomic expression
  - financial predictions - Monte Carlo methods to (eg.) price derivatives
  - run large number of independent problems
- Simulations
  - physical phenomena
  - physics-based codes
  - behavior of physical entities in space and time
  - run single large problem with high interaction between parts

Scientific Simulations

Investigators developing 3D seismic earthquake models of geologically complex Greater Los Angeles Basin.

These models include spatial scales from 10 meters to 100 km, and temporal scales from hundredths of a second to hundreds of seconds, based on highly complex soil properties and geological structures that can only be observed indirectly.

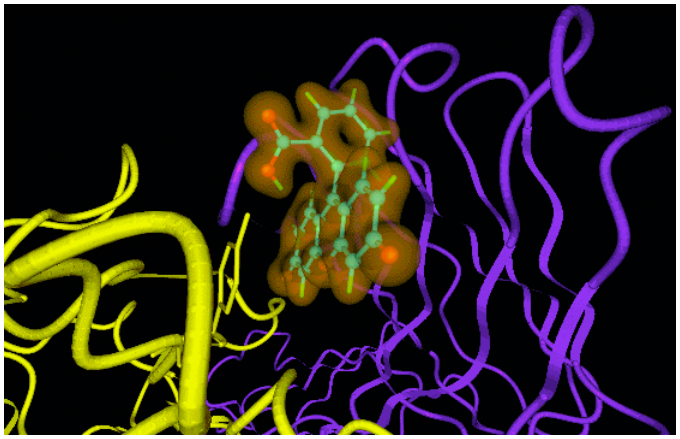
Goal:To simulate a magnitude 7.7 earthquake centered over a 230 km portion of the San Andreas fault at much higher resolution than has previously been performed.



Simulation codes

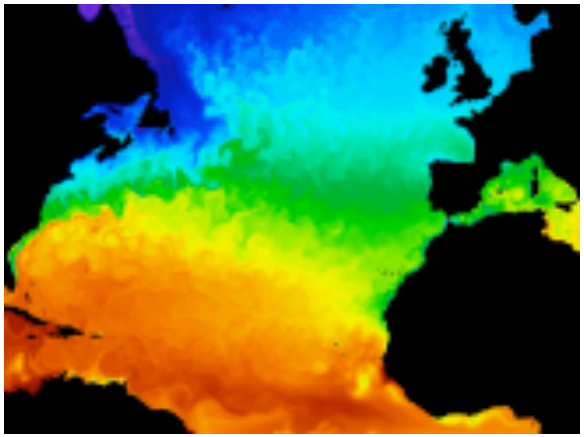
- MILC
  - studies of the mass spectrum of strongly interacting subatomic particles, the weak interactions of these particles, and the behavior of strongly interacting matter under extreme conditions (quantum chromodynamics). **57,000 lines of C code** counting header files and comments.
- GAMESS
  - ab initio studies of quantum chemistry. Computes molecular geometries, energies, etc. **> 100,000 lines of Fortran**.

GAMESS: Flexibility and Molecular Recognition in the Immune System



Simulations

- Parallel Ocean Program (POP)
  - ocean circulation model used as the ocean component of a system climate model; used to resolve eddies in global ocean and ocean-ice models. almost **48,000 lines of Fortran 90**.



- GROMACS
  - molecular dynamics simulations of complex biomolecules; simulates the Newtonian equations of motion upon large systems of particles (i.e. ensemble molecular dynamics). **316,000 lines of C code**.

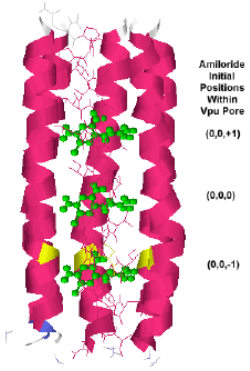
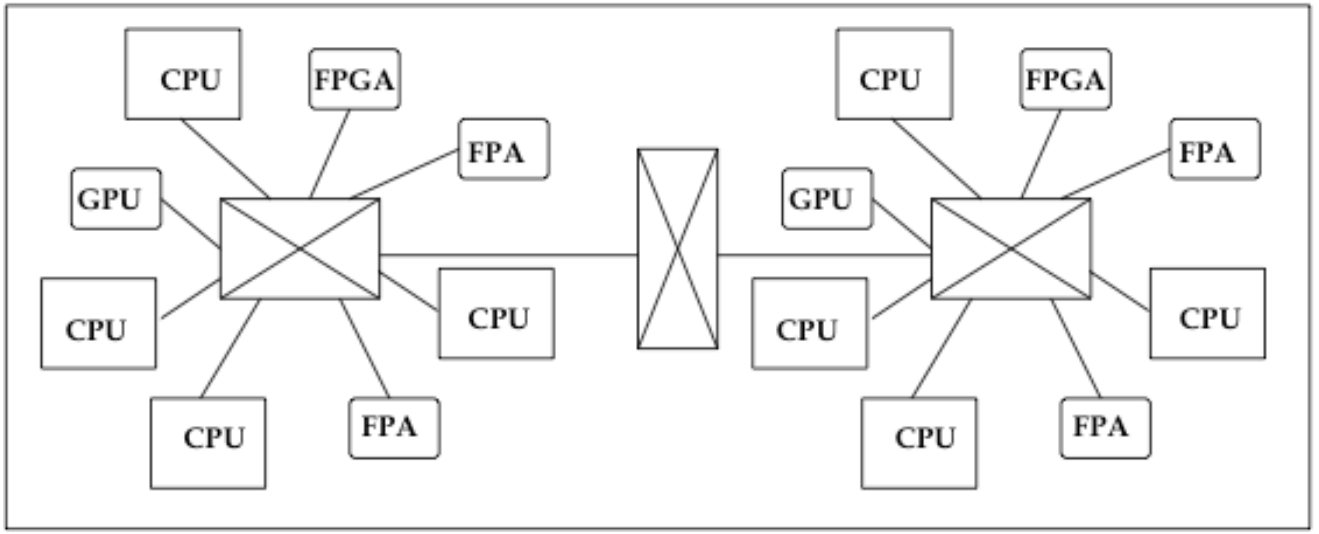


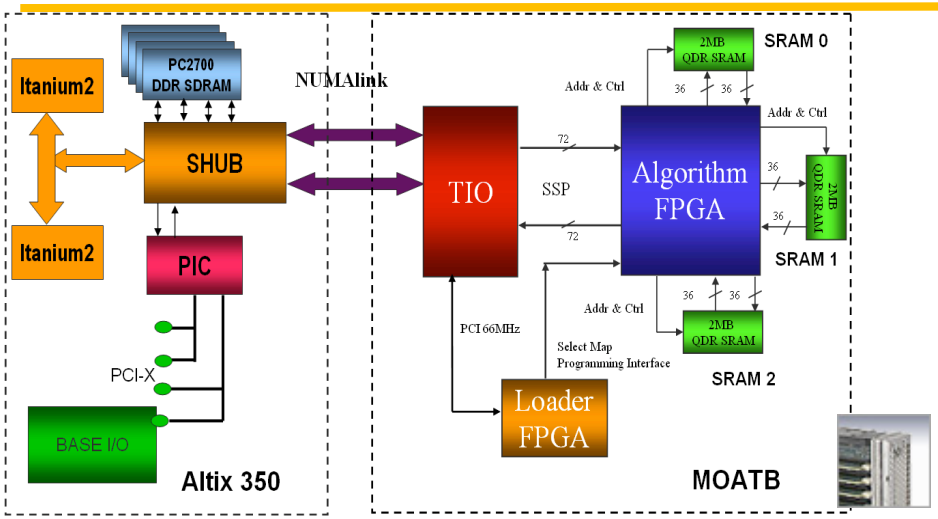
Figure 3.2: A cross-section of the Vpu hexamer, with the N-terminus at the top, showing the three starting positions selected for the amiloride (green). The drug molecules begin each docking simulation in an orientation parallel to the bilayer. The Ser24 residue is highlighted in yellow. One helix has been rendered in wireframe for clarity.

Heterogeneous supercomputers

- High performance, multi-level interconnect
  - Infiniband, Myrinet, GigE
- 64-bit microprocessors
  - multi-core, multi-socket
- co-processors
  - Graphics boards, floating point arrays, FPGAs



SGI RASC



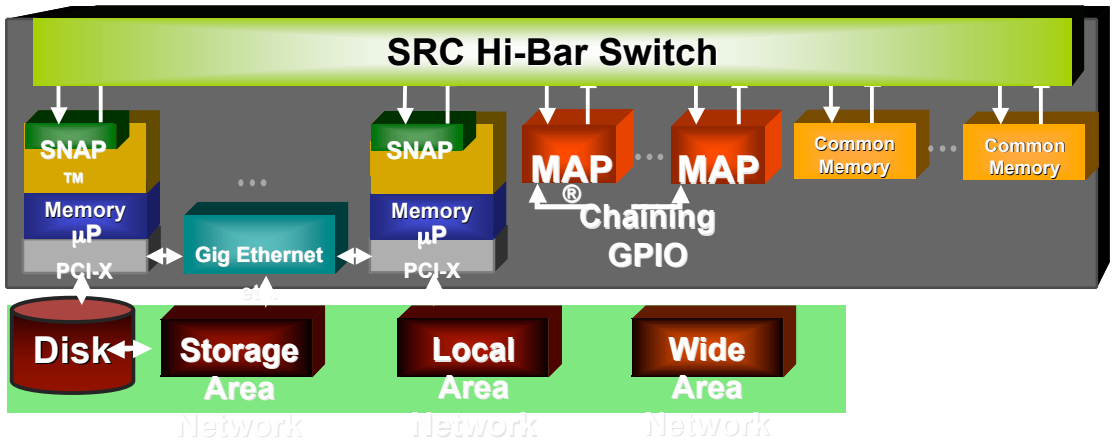
- FPGA board is on NUMALink
- FPGA node is peer to microprocessor
- architecture is well suited to processing data acquisition streams

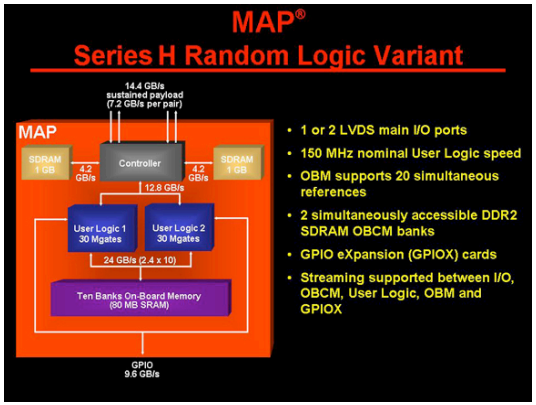
Dual Virtex 4 LX200 FPGAs·
80MB QDR SRAM or 20GB DDR2 SDRAM

Blade or rack-mountable form factor·

Dual NUMAlink - 4 ports

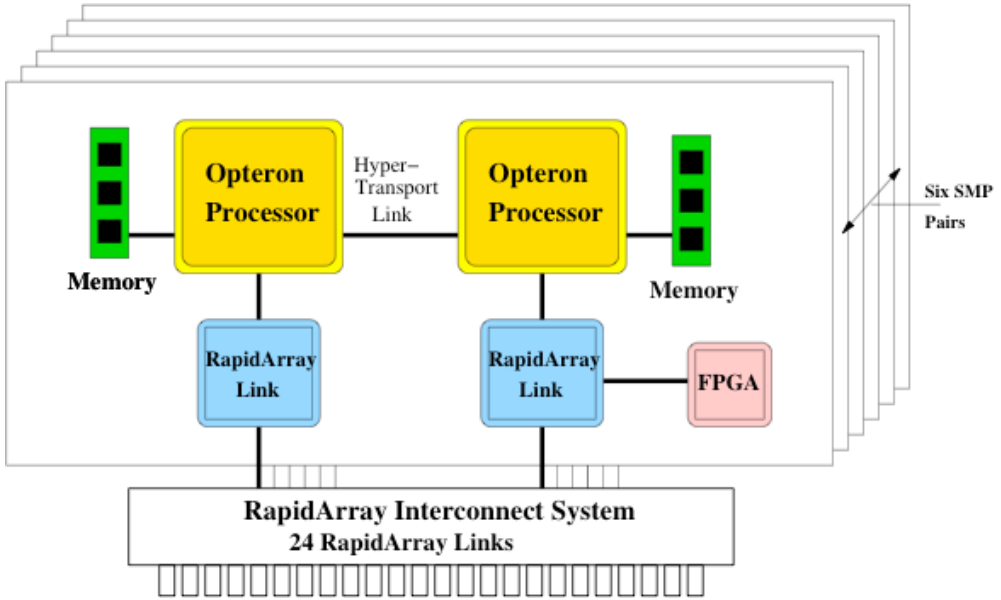
SRC Computer

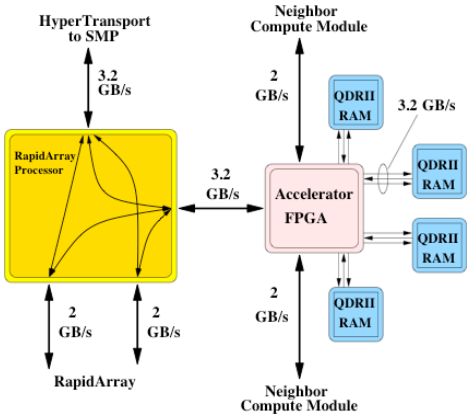




- FPGA board augments microprocessor
- "MAP" on DIMM interface 2.8 GB/s
- 2 largeFPGAs
- multiple banks of on board SRAM
- on board DRAM
- provides for 20 simultaneous memory accesses @ 150MHZ
- architecture supports both library and computational kernel modes

Cray XD1



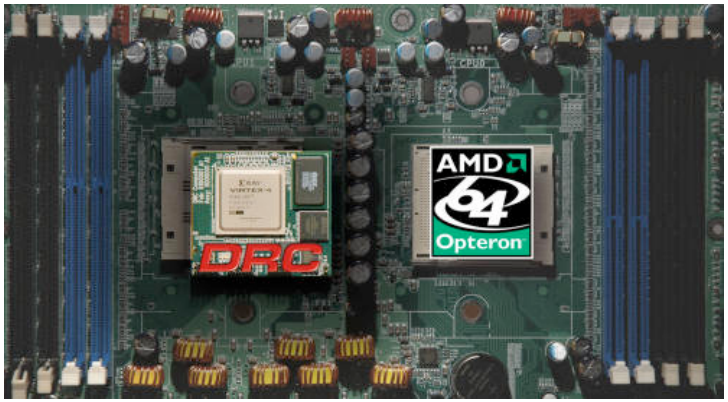


- hypertransport link to Opteron
- Relatively small FPGA
- 16MB QDR SRAM - 12 GB/s BW
- No longer supported by Cray

FPGA co-processors

- Opteron motherboards
- Hypertransport connection between Opteron and FPGA
- Use DIMM slots on MB for FPGA memory
- Include additional off-chip SRAM
- Two companies
  - DRC, XtremeData

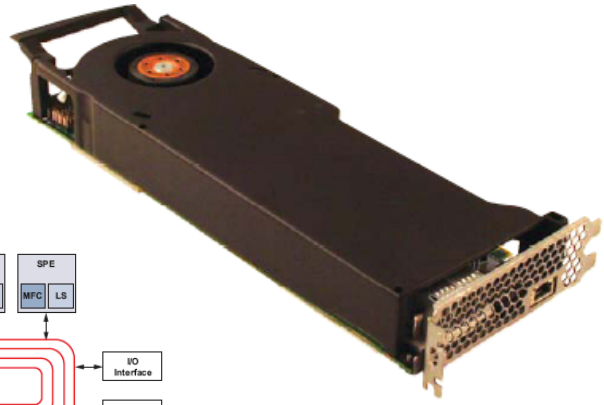
DRC →

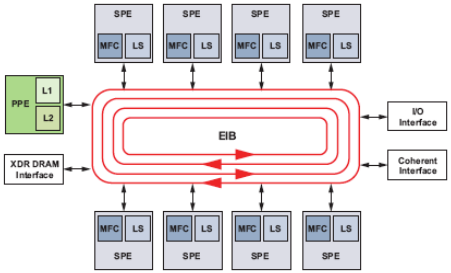


XtremeData ↓

Clusters augmented with Floating Point Arrays

- Clearspeed
  - recently partnered with IBM to build cluster of FPA-accelerated nodes
  - Board contains two CS processors
  - Each processor has 96 double precision SIMD PEs; RISC control processor; I/O controller
  - advertise 50GF sustained DGEMM using 25W
- GPGPU
  - programmable graphics processors
- Cell
  - Mercury and IBM have partnered on Cell Blade architecture
  - “Power” processing element for serial/control
  - 8 SPEs, each can do vectors of 4 single precision FP operations





Problem Statement

- The path to continued performance appears to require heterogeneous computing. We want to assess
- whether a specific large scientific simulation can be accelerated by co-processors? and ...
- if so, what parts of the code should be ported to the co-processor
- need at least 2X speedup to make it worthwhile
- study NSF HPC Scientific Grand Challenge problems
  - work done by Chris Rickett and Chung-Hsing Hsu of LANL

Locating acceleration regions of code

- Study execution profile
  - oprofile, TAU
  - quantify time spent at loop or even line granularity
  - find representative data sets
    - execution profile may vary greatly depending on data set
  - want 80% time in a small region, but that doesn't occur too often
- Study code of likely acceleration candidates
  - data type - integer, single precision FP, double FP
  - types of operations - divides, transcendental functions
  - numbers of operations - how many FP units are needed
  - dependency graph
- Study data profile
  - data consumed and produced in a region must be communicated between global microprocessor memory and FPGA board memory
  - need to know amount of data transferred (per loop iteration)
  - need to know if communication and computation can be overlapped
- We look first for library acceleration opportunities, next for compute kernels

TAU

- TAU automatically inserts timer calls at the start and end of a function
- The instrumented source then gets compiled into the resulting object.
- At run time, each timer is started at the beginning of a routine and stopped at any point of return for the routine.
- TAU gives
  - the total time spent in each routine, and
  - call-path information to show what execution path(s) caused what amount of run time.

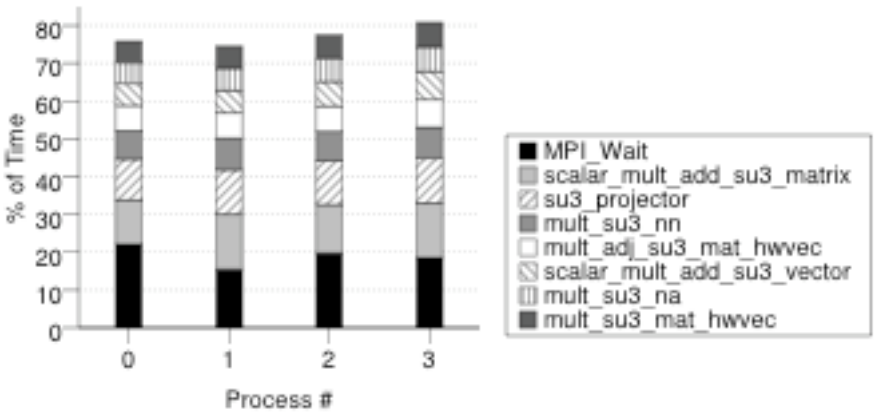
LINPACK

- most popular benchmark, uses linear algebra library, especially DGEMM (double precision, dense matrix multiply)
- 74% time spent in DGEMM
- 5X DGEMM acceleration gives 2.45X overall speedup
- 10X DGEMM acceleration gives 3X overall speedup

Processor	Performance (GF/s)
3.4GHz Pentium 4 Prescott	**5.61**
PPC G5 with Altivec	**10**
Cell	**14.6**
V2-6K	**12**
Clearspeed CS600	**25**

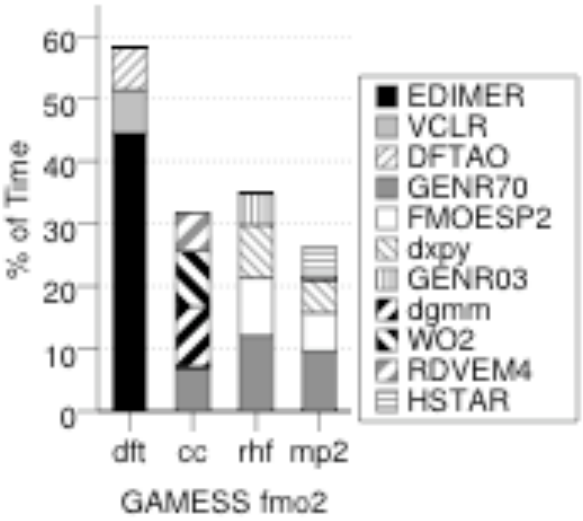
MILC profile

- timing results from running on a dual socket 2.4 GHz Xeon (2GB memory, icc/icpc 8.0, ifort 9)
- up to 20% time in MPI_Wait
- 55% in matrix algebra routines other than MPI_Wait and taking > 4% run time
- ignoring data transfer time, speedup of 8X needed in those routines to get 2X speedup on the application



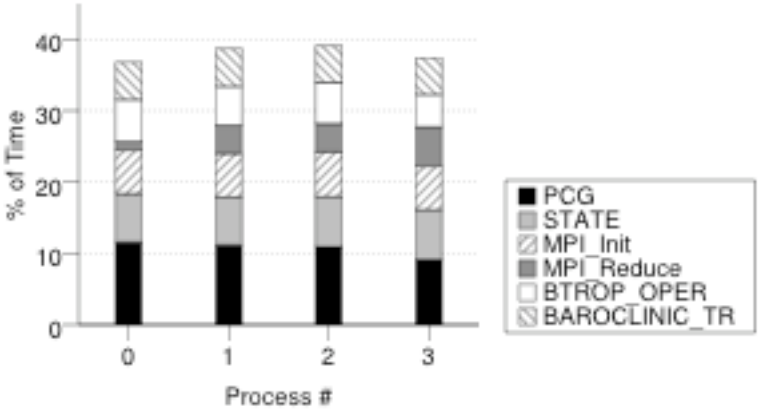
GAMESS profile

- Largest time spent in EDIMER for one data set
  - long, complex routine with lots of control flow
  - not immediately amenable to acceleration
- dxpy and dgmm (wrappers to DAXPY and DGEMM) account for 10%
- not enough library routine usage to benefit from library-based acceleration
- must examine other routines for possible compute kernel acceleration



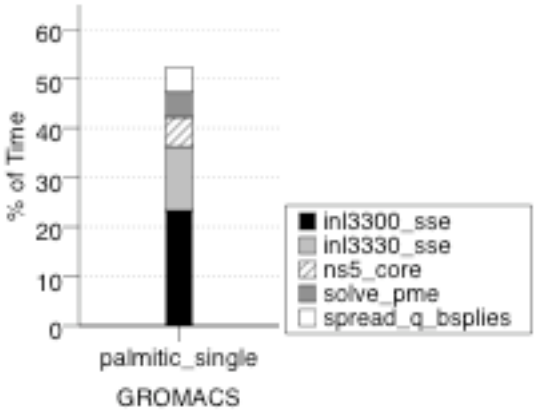
POP profile

- Top library routine is conjugate gradient solver (PCG)
- average 10.7% of the run time
- max speedup of 12%
- Hardware PCG library would have little impact on POP run time.
- combination of all kernels < 30%
- poor candidate for co-processor acceleration

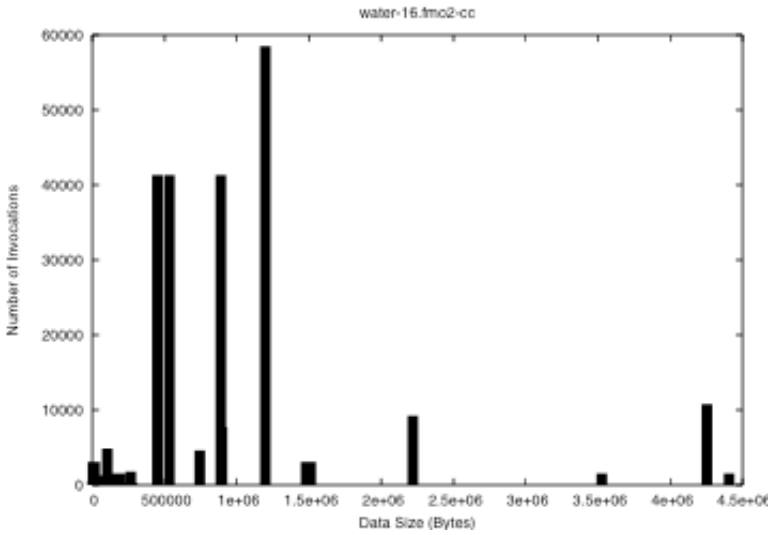


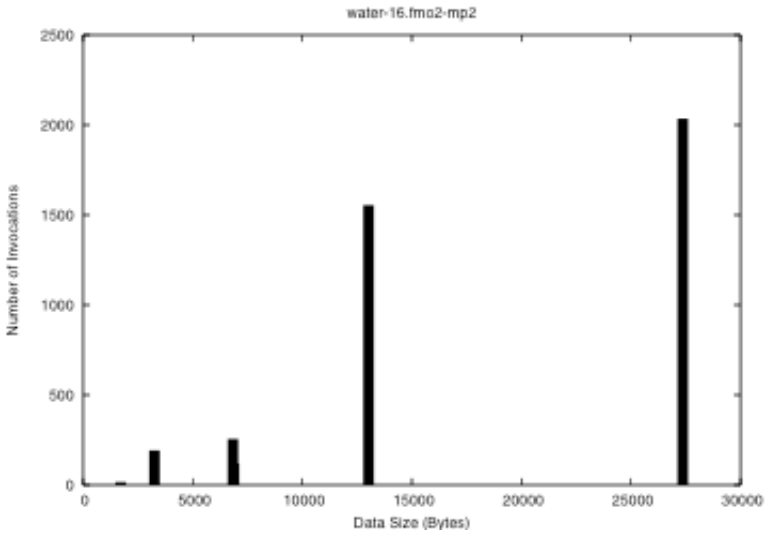
GROMACS Profile

- profiled with gprof on Xeon for a palmitic acid simulation
- no library routines used
- specially coded sse routines used - inl3300, inl330
- up to 55% in 5 routines
  - solve_pme very complex
  - sse routine already highly optimized, so speedup using co-processor will likely be less than when compared to Xeon routine



Data Profile





GAMESS: DGEMM array sizes

- vary with the data sets
  - lots of calls with relatively small size arrays
  - lots of calls with large array sizes

No easy answers

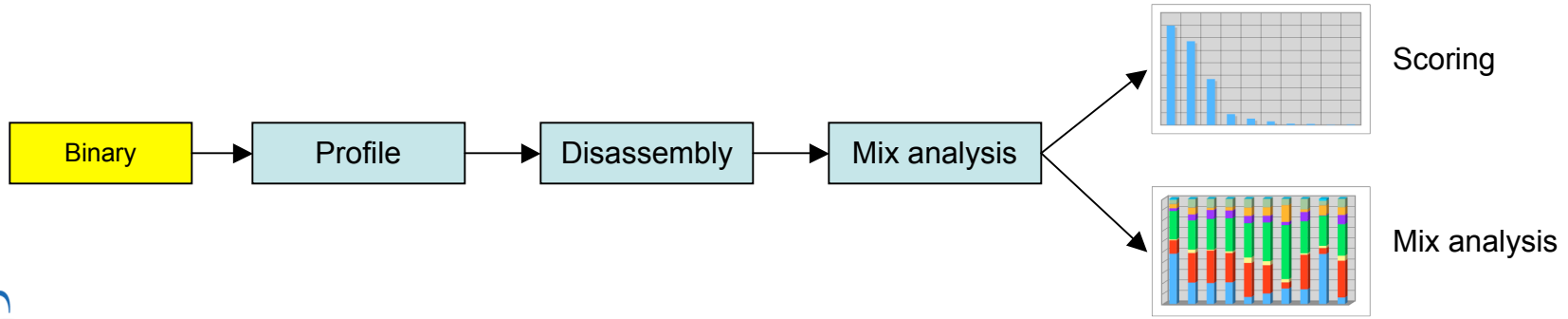
- Acceleration using only library routines will be negligible for scientific codes.
  - Immediate impact only on Linpack
- It will be necessary to accelerate compute kernels in the applications
  - some of them are long and complex
  - need to understand static and dynamic behavior
- Must also re-think application's algorithms and data access patterns
- Need software analysis tools to help understand applications and their potential for co-processor acceleration

Automated static and dynamic analysis

- profiling to line level of granularity
- collect high computational density kernels
- analyze data types and operations used in the kernels
- estimate memory footprint of kernels
- estimate performance of partitioned application
  - kernel performance, communication cost, time spent in remaining sequential code

Tool overview

- The OpMix tool is a two-phase process:
  - First, profiling is performed to identify prime candidates for acceleration.
    - Sequential codes: `gprof` or TAU
      - TAU provides loop-level profiling.
    - Parallel codes: TAU
  - Second, the top N functions are examined in compiled form to analyze their instruction mix.
    - In the results shown in this talk, N=10.
- Analysis for each routine is emitted as a breakdown by percentage in each instruction class, and a single scalar “goodness” score.



Instruction mix scoring

- For each routine, the tool emits the percentage of instructions in each class (P_{class}), and the overall score.
- Each class is given a weight (W_{class}).
- The overall score for a given routine is computed as:

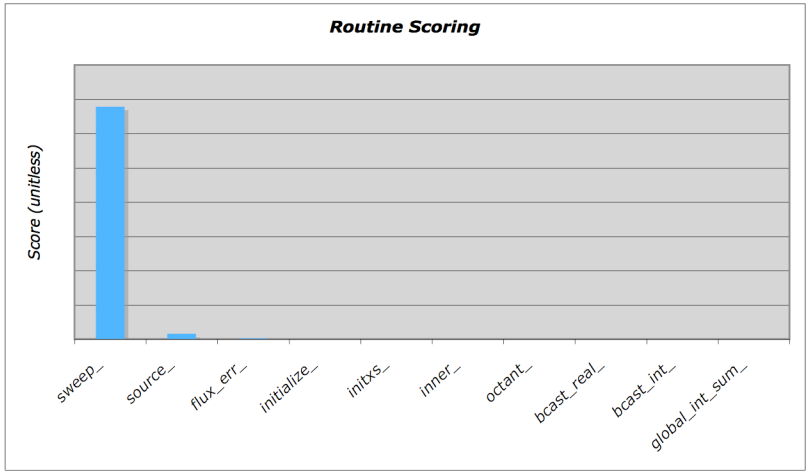
$$score = \frac{T_{routine}}{T_{total}} \sum_{classes} W_{class} P_{class}$$

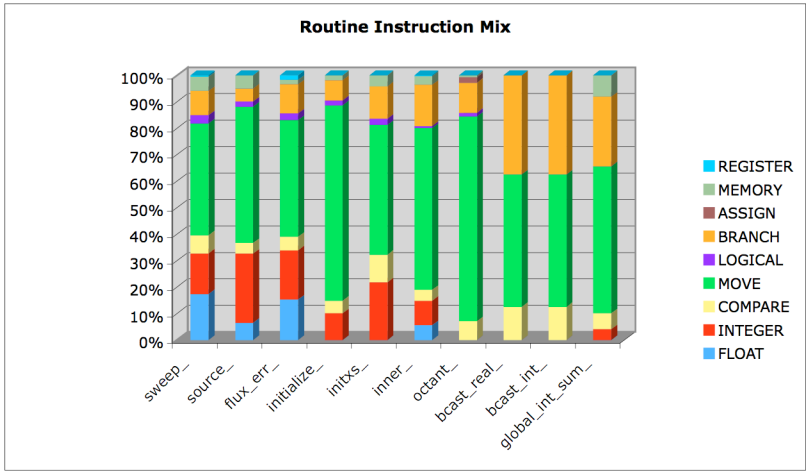
Class weighting

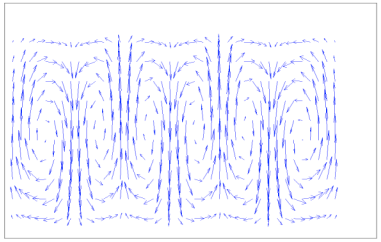
- Classes are weighted based on the relative appropriateness of each class for a given target.
- For example, a target that is highly capable for integer and logical operations, moderately capable for floating point, and poorly capable for branches and memory references could be weighted as:
  - $W_{logical} = 1.0$
  - $W_{integer} = 0.9$
  - $W_{float} = 0.5$
  - $W_{branch} = 0.25$
  - $W_{mem} = 0.1$
- A single profile and binary can be evaluated for multiple targets by providing different weight sets for each.
- Clearly this is very simplistic. A better score will include memory coverage information and more sophisticated target constraints with respect to code structure.
  - The scoring shown here suffices for our purposes.

Sweep3D

- The Sweep3D code spends nearly all time in the `sweep_` function. As we can see, only one other function has a non-negligible time profile and instruction mix score.
- This example requires finer-grained profiling at the loop or statement level to identify regions of code to perform mix analysis upon.
  - Profiling is the hard part. The current mix analysis tool supports arbitrary byte streams for disassembly and analysis once a profile identifies regions.

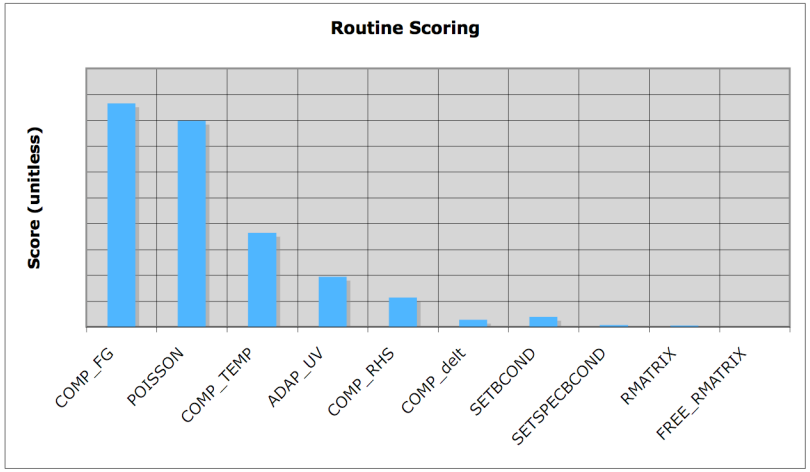


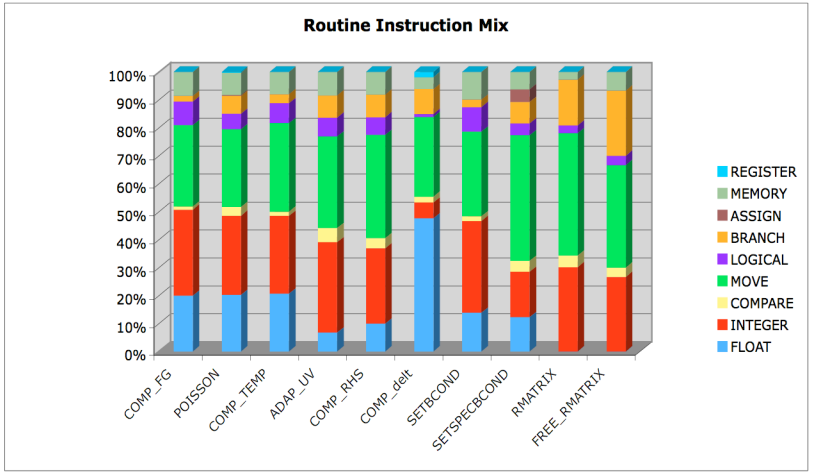




CFD: Rayleigh-Taylor

- The CFD code under the Rayleigh-Taylor input shows three or four functions that are candidates for acceleration based on their instruction mix and profile time.
- `COMP_FG` has a comparable time profile as `POISSON`, but exhibits a lower percentage of branching instructions.

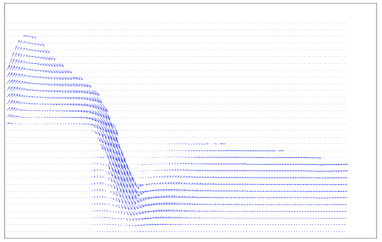




Los Alamos NATIONAL LABORATORY EST. 1943

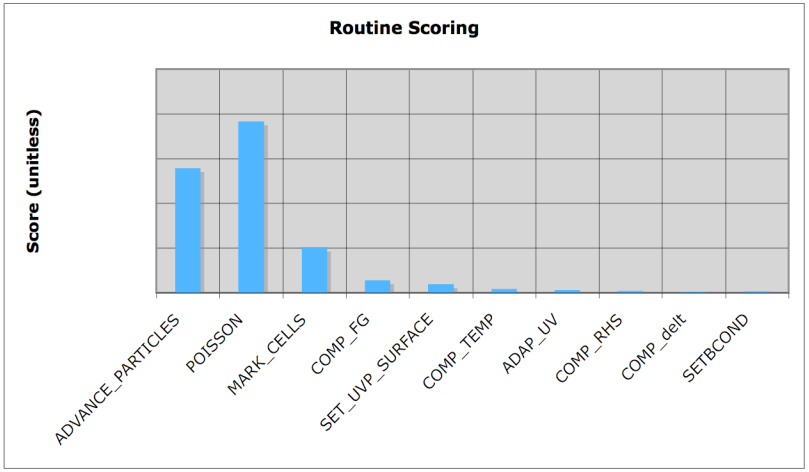
Operated by the Los Alamos National Security, LLC for the DOE/NNSA

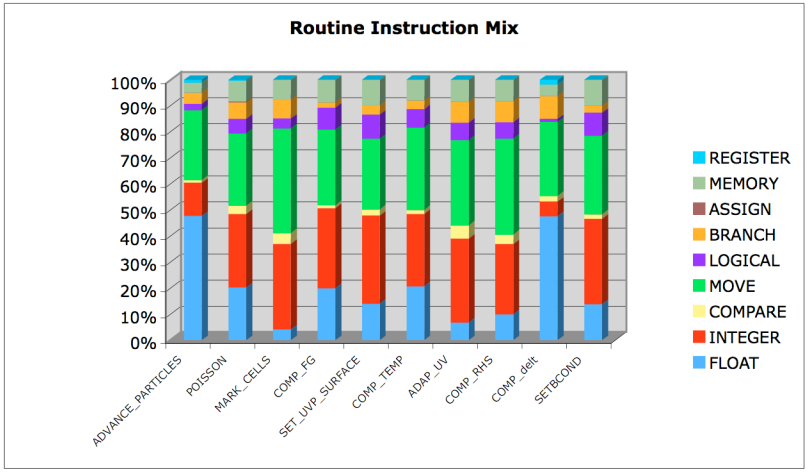
NNSA

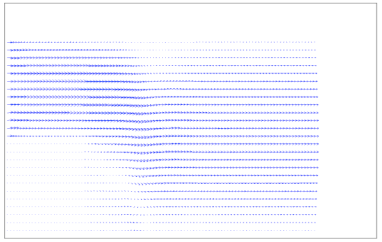


CFD: Wave

- The `ADVANCE_PARTICLES` routine is the most time consuming by profiling alone.
- After instruction mix analysis and scoring, we can see that the next most time consuming routine, `POISSON`, is likely to have a better payoff in performance.
- Observe that this is because the `ADVANCE_PARTICLES` routine has a higher percentage of floating point instructions which are undesirable relative to integer operations.

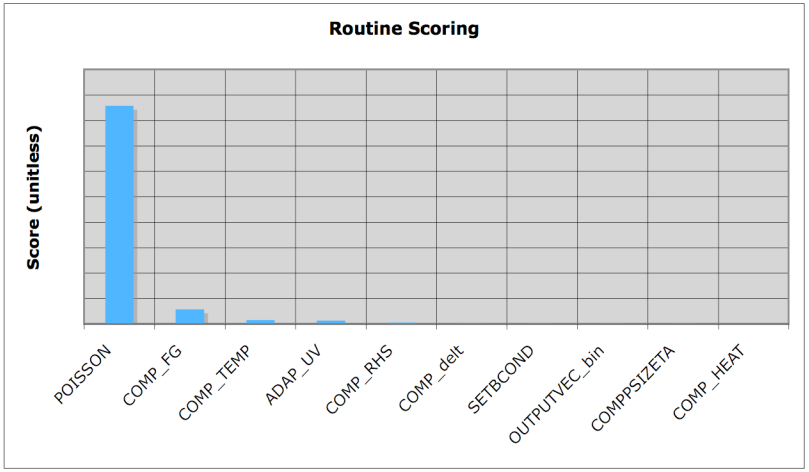


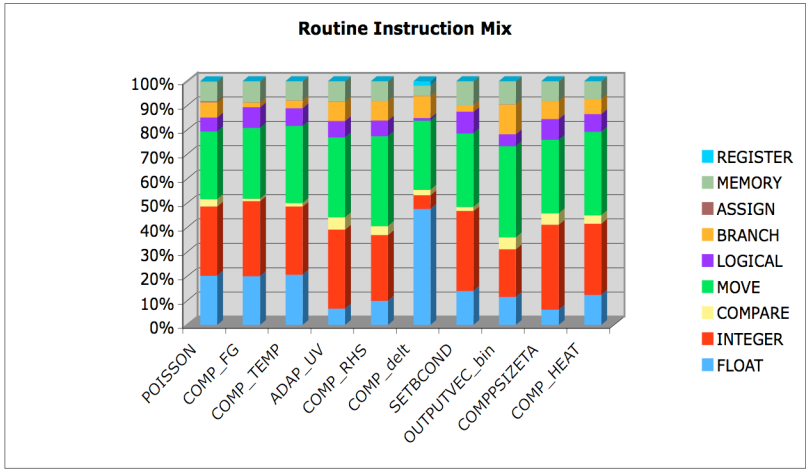


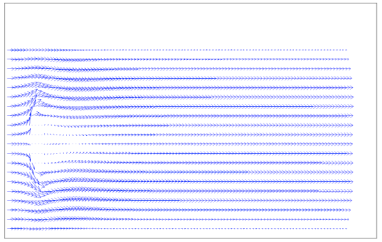


CFD: Backstep

- The backstep input exercises a single hotspot.
- As we can see, this technique reveals the clear candidate for acceleration (`POISSON`).
- If we used this input alone though, we would miss the fact that other candidates are equally important for general inputs.
  - Previous slides show this.

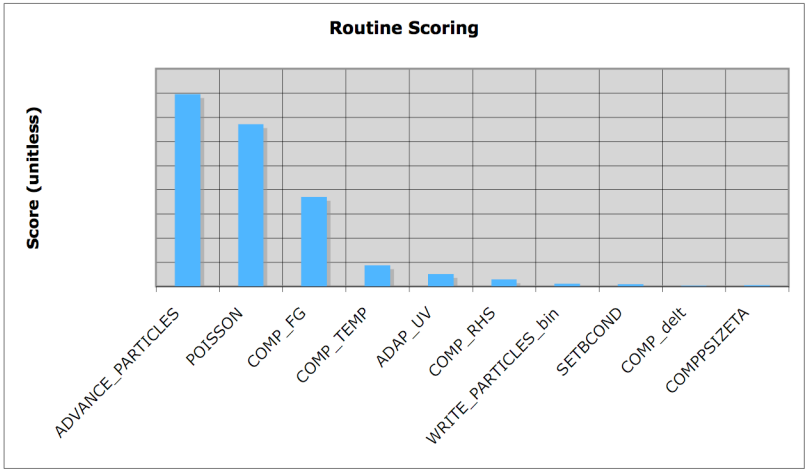


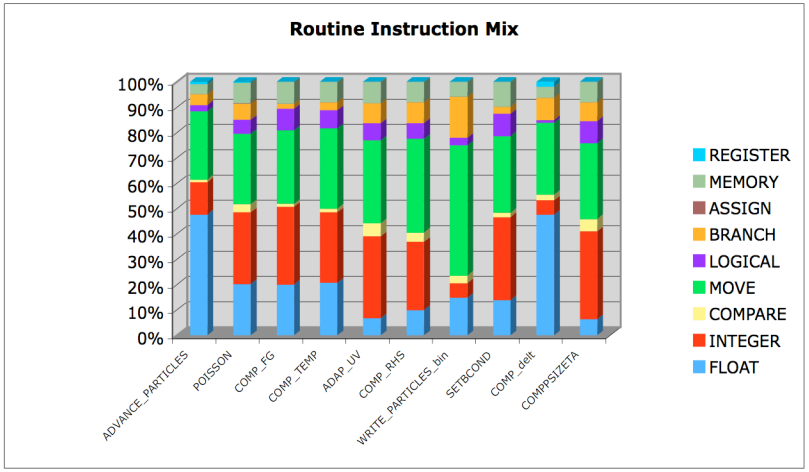




CFD: Obstacle

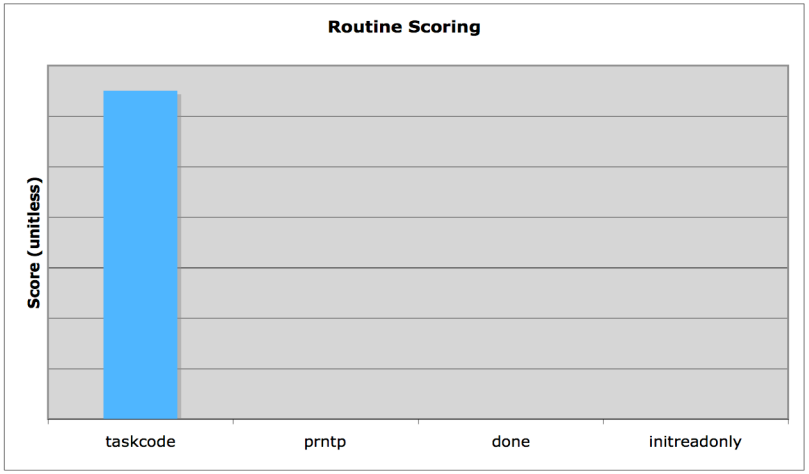
- The obstacle input shows a similar profile to rayleigh, but `POISSON` switches with `ADVANCE_PARTICLES`.
- This indicates that both routines should be examined for acceleration.
- Notice that `ADVANCE_PARTICLES` is very heavily floating point relative to `POISSON`. The higher score indicates that the higher time for `ADVANCE_PARTICLES` outweighs this mix difference.

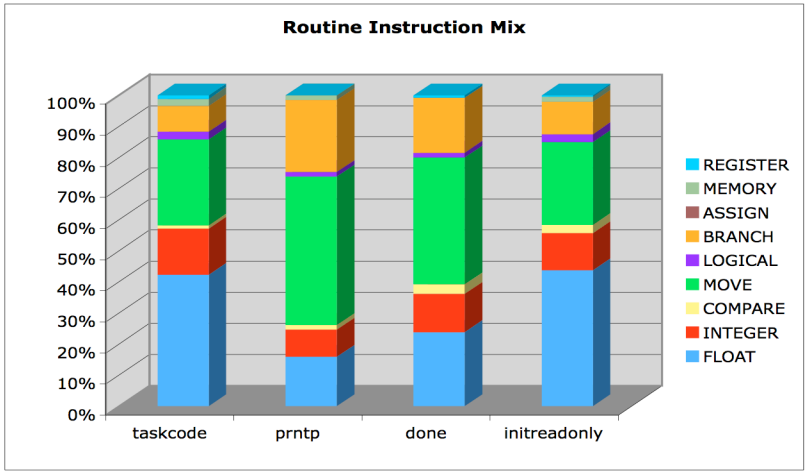




Photon

- The photon code is the simplest, and worst representative for this tool.
- All of the time is spent in a single routine that performs all of the work.
- Deeper analysis requires the `taskcode` routine to be decomposed into regions for finer grained profiling and analysis.





Valgrind

- automatically detect many memory management and threading bugs
- detailed profiling to understand dynamic program behavior
  - identify bottlenecks to speed up code
  - reduce memory use
- includes
  - memory error detector
  - cache (time) profiler
  - call-graph profiler
  - heap (space) profiler
- runs on X86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux

- Translates x86/PPC binary into a virtual machine
- Interprets the virtual machine code
- Tool developer can insert instrumentation as C code

✓ can gain visibility into cache behavior, memory addresses, dynamic memory management

– slows down execution by factor of 10

– has trouble dealing with threaded code

Memfoot Memory Footprint Tool

- builds on Valgrind “lackey” tool
- records number of **unique** memory addresses accessed
- records number of times each memory address is accessed
- profile is collected on subroutine granularity
- subroutine chosen by user after profiling with tau, oprofile, gprof, ...
- usage count helps identify location in memory hierarchy to store data
  - register, local SRAM, global DRAM

Memfoot results

- photon
  - 3682 unique addresses
  - 1,361,580,411 total addresses
  - 3676 addresses: <1% of the time
  - 1 address: 2%
  - 1 address : 5%
  - 1 address : 9%
  - 1 address : 10%
  - 1 address : 13%
  - 1 address : 22%
- sweep
  - 24678 unique addresses
  - 11,956,605,201 total addresses
  - 24660 addresses : <1% of the time
  - 5 addresses : 2%
  - 9 addresses : 3%
  - 1 address : 5%
  - 3 address : 9%

CFD results - different data sets, functions

prg_obstacle_advance_particles 985 calls

(hash table collisions) 24666 addresses: 1 % of the time

3 addresses: 2 %

1 address: 5 %

2 addresses: 6 %

2 addresses: 7%

2 addresses: 9%

1 address: 10%

1 address: 13 %

24,678 Unique addresses, 11,05,396,820 total addresses

prg_obstacle_poisson 985 calls

7723 addresses: 1 %

3 addresses: 3 %

6 addresses: 5 %

1 address: 7 %

7733 Unique addresses, 1,259,813,619 total addresses

prg_rayleigh_comp_fg 2017 calls

3058 addresses: 1 %

9 addresses: 2 %

1 address: 3 %

2 addresses: 4 %

2 addresses: 5 %

3072 Unique addresses, 93,754,194 total addresses

prg_rayleigh_poisson 2017 calls

1636 addresses: 1 %

4 addresses: 2 %

5 addresses: 4 %

1 address: 5 %

1 address: 8 %

1647 Unique addresses, 160,337,856 total addresses

Observations, questions, future work ...

- Number of unique addresses gives an idea of the data requirements
  - 5K-25K for these benchmarks
  - fit on-chip for many co-processors
- Number of unique addresses gives some hint of I/O bandwidth needed between processor and co-processor
  - how many unique accesses per call?
- Number of times accessed suggests location in memory hierarchy
  - >2% of the total accesses ➞ keep in registers
- How does memory footprint change as problem scales?
  - need to run bigger codes with more data
  - valgrind scaling issues ...
- Currently tool operates on subroutine granularity
  - loop level memory footprint would be useful for some codes

Conclusions

- Co-processor parallelism is asymmetric
- scientific applications must be analyzed from a different perspective than for traditional cluster-based parallel processing
- Difficult to find heavy library usage
- Difficult to find sufficiently compute intensive kernels
- Tools are needed to help understand kernels at the detailed level
- Tools development in progress at LANL
  - study mix of operations in subroutine of interest
  - study memory access footprint and counts