



Reduced-Precision Floating-Point Analysis via Binary Modification

Mike Lam, UMD

Jeff Hollingsworth, UMD

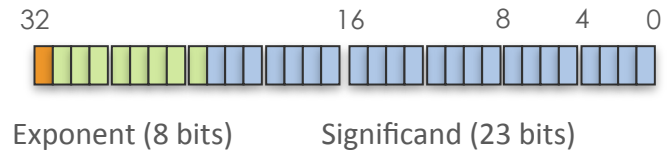
Context

Floating-point arithmetic represents real numbers as

$$(\pm 1.\mathit{frac} \times 2^{\mathit{exp}})$$

- – Sign bit
- – Exponent
- – Significand (“mantissa” or “fraction”)

Single Precision

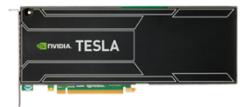


Double Precision



Context

- Floating-point is ubiquitous **but problematic**
 - Rounding error
 - Accumulates after many operations
 - Not always intuitive (e.g., non-associative)
 - Naïve approach: higher precision
 - Lower precision is preferable
 - Tesla K20X is 2.3X faster in single precision
 - Xeon Phi is 2.0X faster in single precision
 - Single precision uses 50% of the memory bandwidth



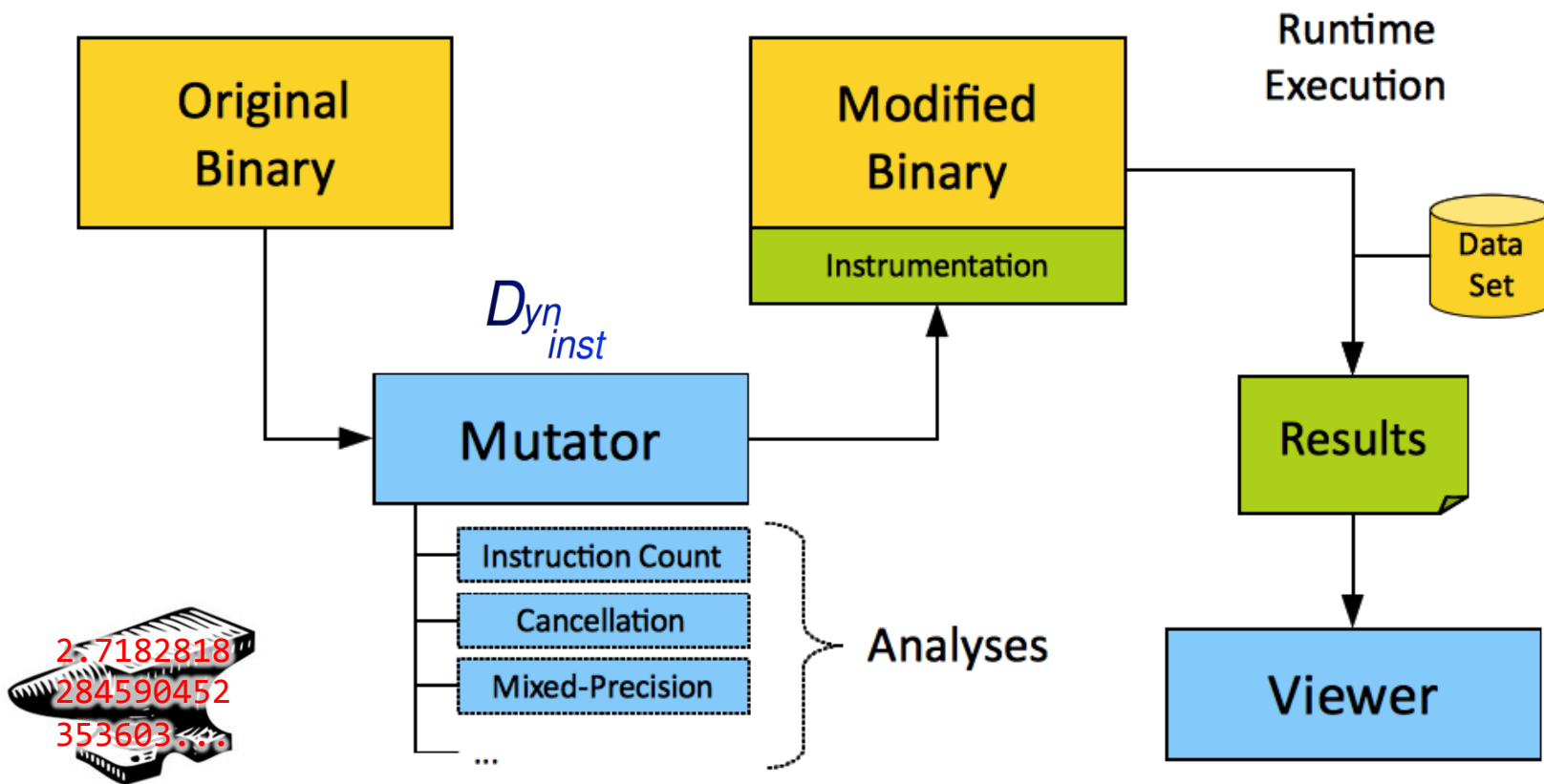
Research Contributions

- Software framework (CRAFT)
- Previous work
 - Cancellation detection [PARCO2012]
 - Mixed-precision configuration [ICS'13]
- Recent work
 - Reduced-precision analysis
- Future work



Framework

CRAFT: Configurable Runtime Analysis for Floating-point Tuning



Framework

- Dyninst: a binary analysis library
 - Parses executable files (InstructionAPI & ParseAPI)
 - Inserts instrumentation (DyninstAPI)
 - Supports full binary modification (PatchAPI)
 - Rewrites binary executable files (SymtabAPI)
- XED instruction decoder (from Intel's Pin)

*Dyn
inst*

Framework

- CRAFT framework
 - Dyninst-based binary mutator (C/C++)
 - Swing-based GUI viewers (Java)
 - Automated search scripts (Ruby)
 - Over 30K LOC total
 - LGPL on Sourceforge: sf.net/p/crafthpc

Cancellation Detection

- Loss of significant digits due to subtraction

$$\begin{array}{r} 2.491264 \quad (7) \\ - 2.491252 \quad (7) \\ \hline 0.000012 \quad (2) \end{array} \qquad \begin{array}{r} 1.613647 \quad (7) \\ - 1.613647 \quad (7) \\ \hline 0.000000 \quad (0) \end{array}$$

(5 digits cancelled)

(all digits cancelled)

- Cancellation detection
 - Instrument every addition and subtraction
 - Report cancellation events

Mixed Precision

- Frequent operations use single precision
- Crucial operations use double precision

- 1: $LU \leftarrow PA$
- 2: solve $Ly = Pb$
- 3: solve $Ux_0 = y$
- 4: for $k = 1, 2, \dots$ do
- 5: $r_k \leftarrow b - Ax_{k-1}$**
- 6: solve $Ly = Pr_k$
- 7: solve $Uz_k = y$
- 8: $x_k \leftarrow x_{k-1} + z_k$**
- 9: check for convergence
- 10: end for

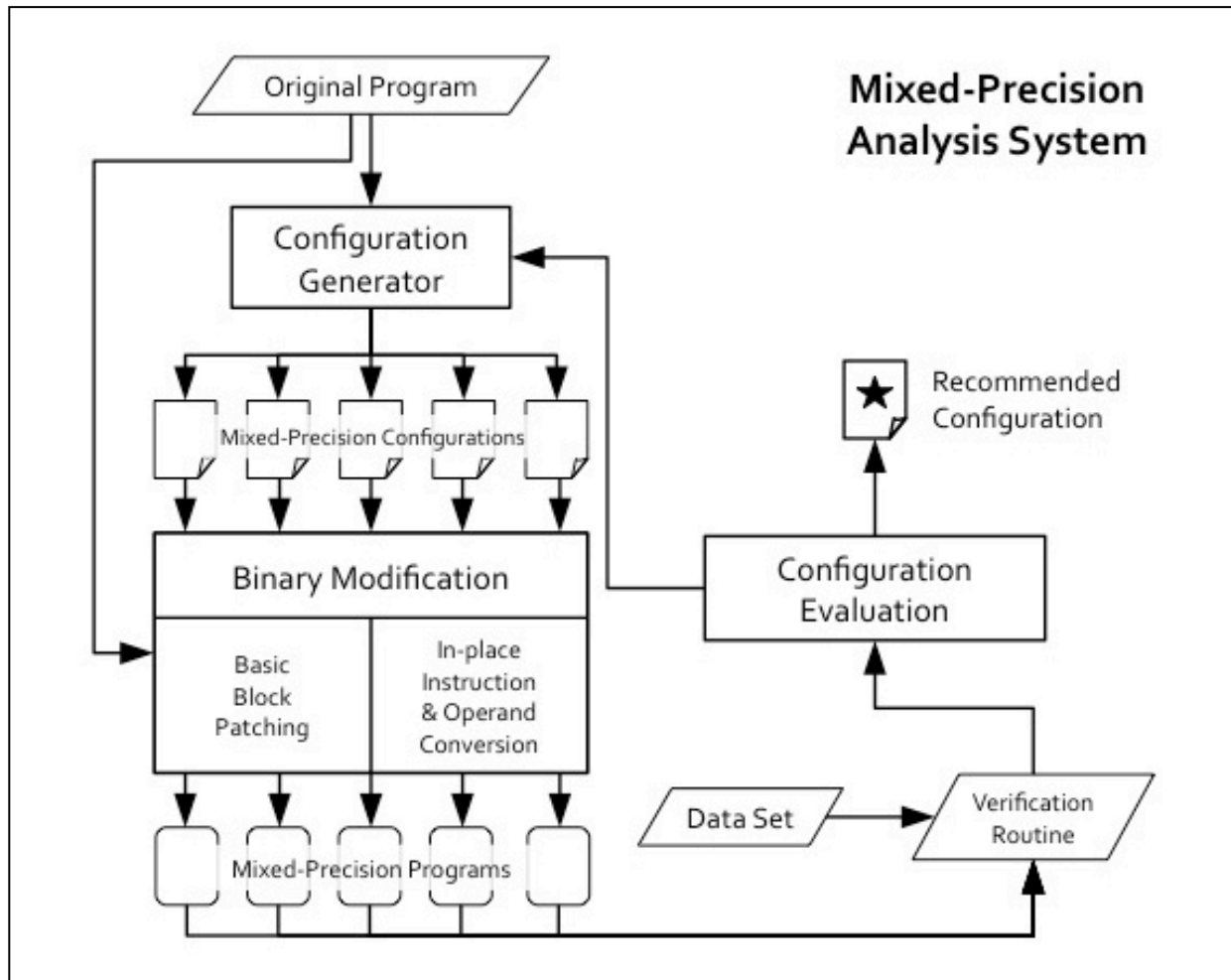
**Mixed-precision
linear solver**

[Buttari 2008]

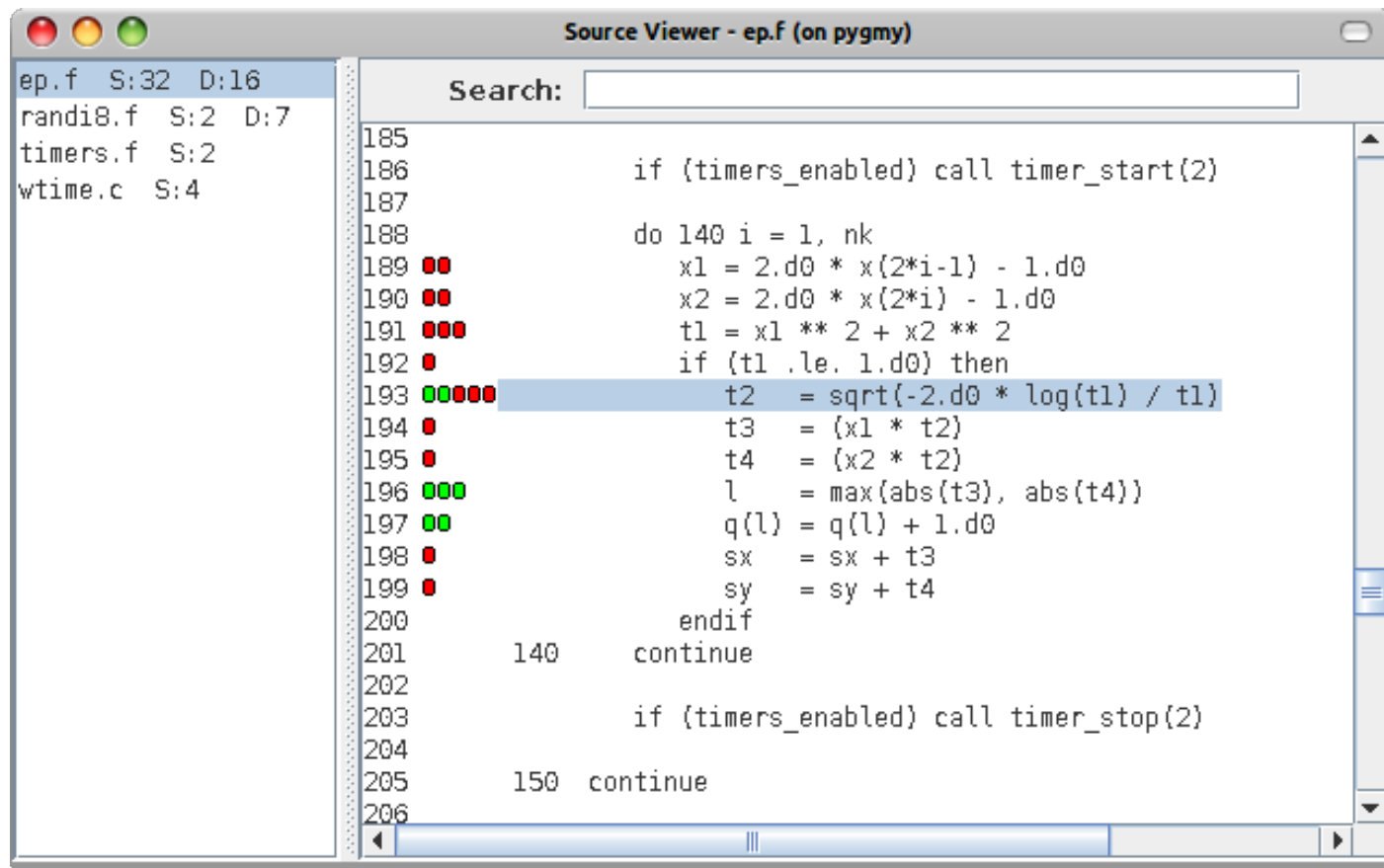
**Red text indicates
double-precision
(all other steps are
single-precision)**

50% speedup on average
(12X in special cases)

Automated Search



Automated Search



The screenshot shows a Source Viewer window titled "Source Viewer - ep.f (on pygmy)". The window is divided into two panes. The left pane shows a file list with the following entries: ep.f S:32 D:16, randi8.f S:2 D:7, timers.f S:2, and wtime.c S:4. The right pane shows the source code for ep.f, with a search bar at the top. The code is as follows:

```
185
186     if (timers_enabled) call timer_start(2)
187
188     do 140 i = 1, nk
189     ●●      x1 = 2.d0 * x(2*i-1) - 1.d0
190     ●●      x2 = 2.d0 * x(2*i) - 1.d0
191     ●●●     t1 = x1 ** 2 + x2 ** 2
192     ●       if (t1 .le. 1.d0) then
193     ●●●●●   t2 = sqrt(-2.d0 * log(t1) / t1)
194     ●       t3 = (x1 * t2)
195     ●       t4 = (x2 * t2)
196     ●●●     l = max(abs(t3), abs(t4))
197     ●●     q(l) = q(l) + 1.d0
198     ●       sx = sx + t3
199     ●       sy = sy + t4
200           endif
201     140     continue
202
203           if (timers_enabled) call timer_stop(2)
204
205     150     continue
206
```

The code is annotated with colored dots indicating execution progress: red dots for lines not yet executed, green dots for lines currently being executed, and blue highlights for lines that have been fully executed. Line 193 is highlighted in blue and has five green dots. Line 192 has one red dot. Lines 189-191 have two red dots. Lines 194-195 have one red dot. Lines 196-197 have three green dots. Lines 198-199 have one red dot. Lines 185-187, 200, 201, 203, 204, and 205 have no dots.

Mixed Precision: Results

Benchmark (name.CLASS)	Candidate Instructions	% Dynamic Replaced
bt.A	6,262	78.6
cg.A	956	5.6
ep.A	423	45.5
ft.A	426	0.2
lu.A	6,014	57.4
mg.A	1,393	36.6
sp.A	4,507	30.5

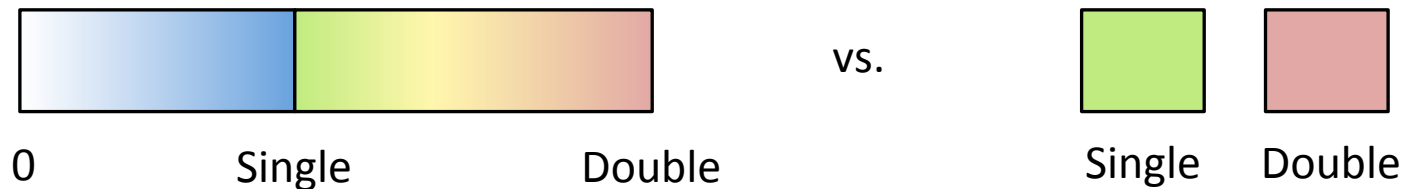


Mixed Precision: Conclusions

- Automated analysis can illuminate cancellation behavior
- Automated search can provide precision-level replacement insights
 - Still very coarse-grained w/ binary decision-making

Reduced Precision

- Simulate reduced precision with truncation
 - Truncate result after every operation
 - Allows zero up to double (64-bit) precision
 - Less overhead (fewer added operations)
- Search routine
 - Identifies component-level precision requirements



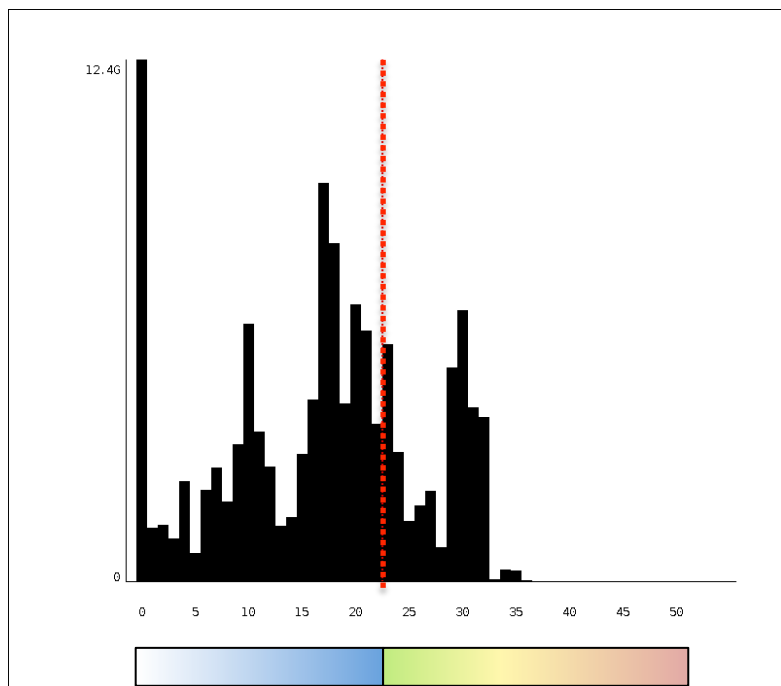
Reduced Precision

- Faster search convergence compared to mixed-precision analysis

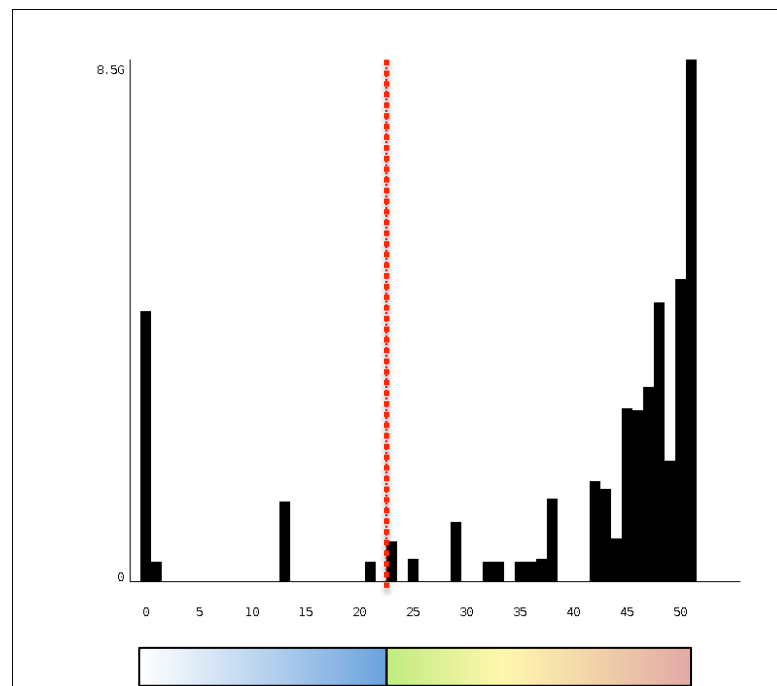
Benchmark	Original Wall time (s)	Speedup
cg.A	1,305	59.2%
ep.A	978	42.5%
ft.A	825	50.2%
lu.A	514,332	86.7%
mg.A	2,898	66.0%
sp.A	422,371	44.1%

Reduced Precision

- General precision requirement profiles

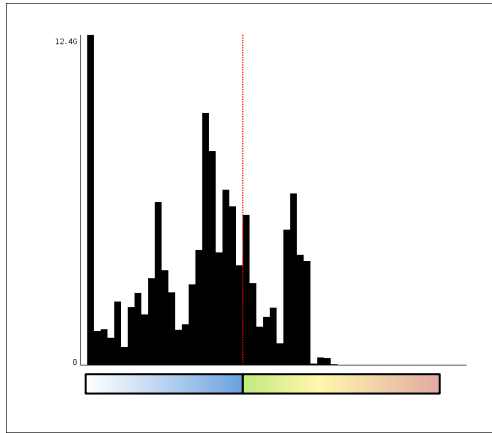


Low sensitivity

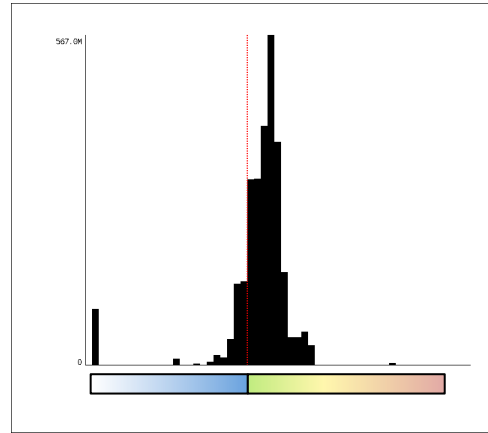


High sensitivity

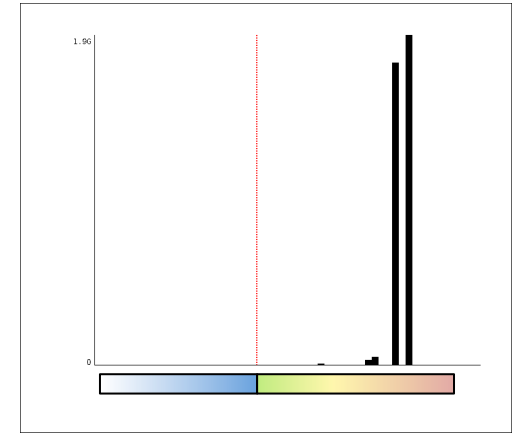
Reduced Precision: Results NAS (top) & LAMMPS (bottom)



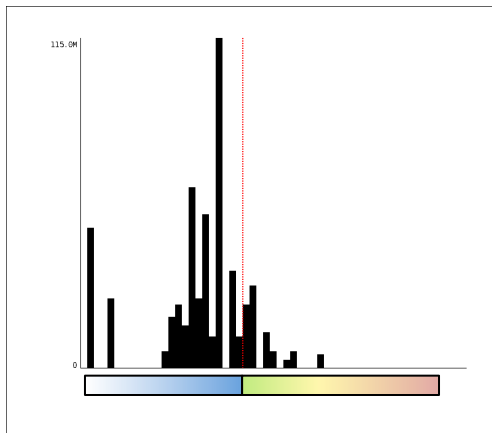
bt.A (78.6%)



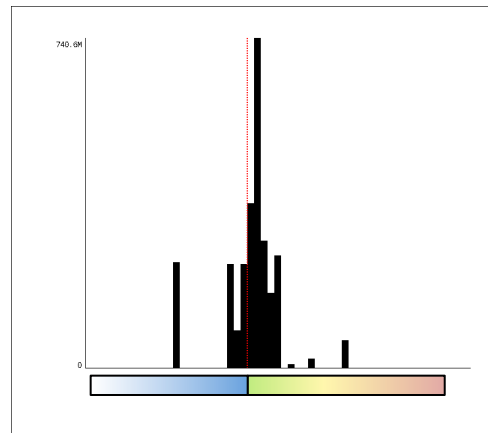
mg.A (36.6%)



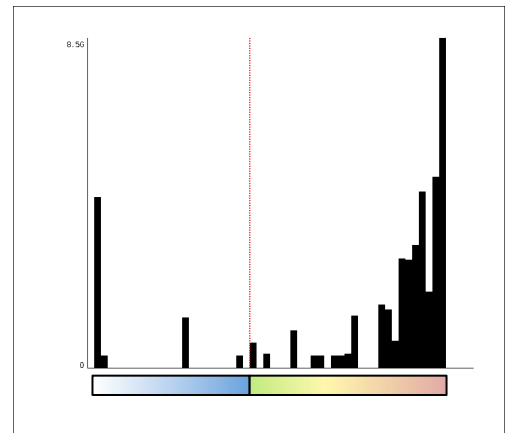
ft.A (0.2%)



chute

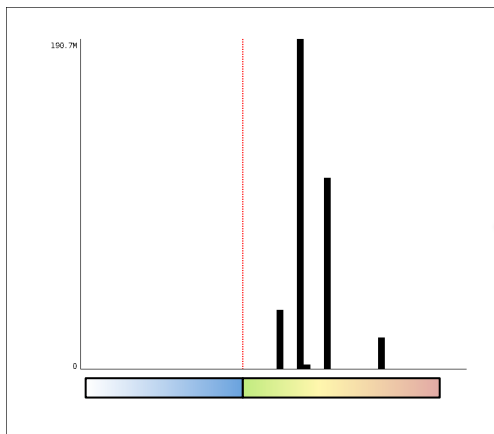


lj

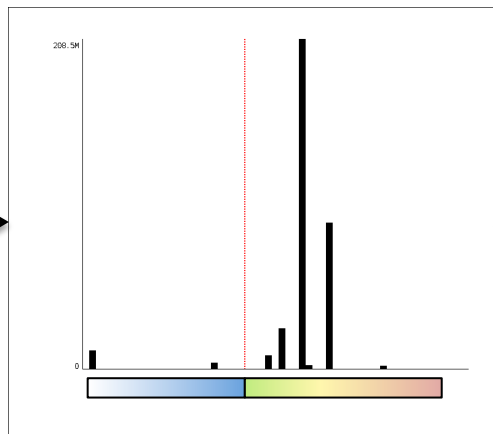


rhodo

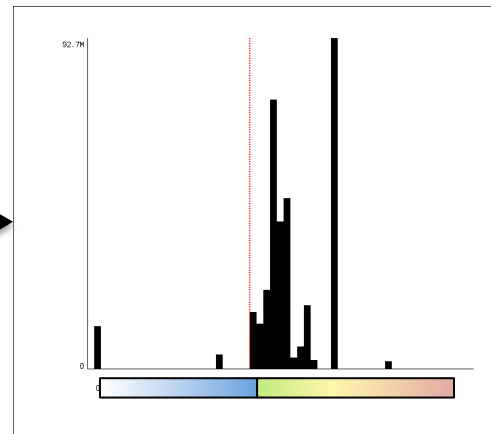
Reduced Precision: Results NAS mg.W (incremental)



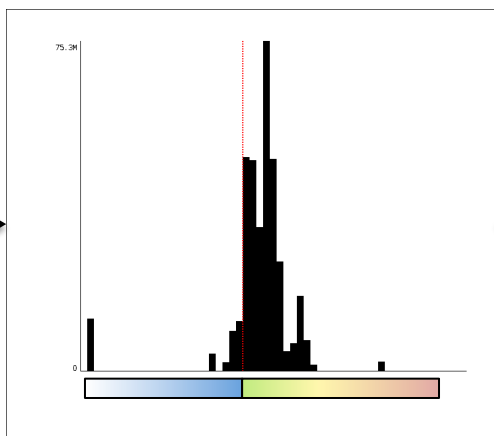
>5.0% - 4:66



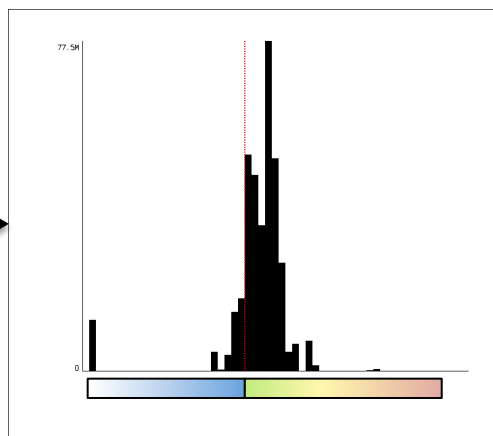
>1.0% - 5:93



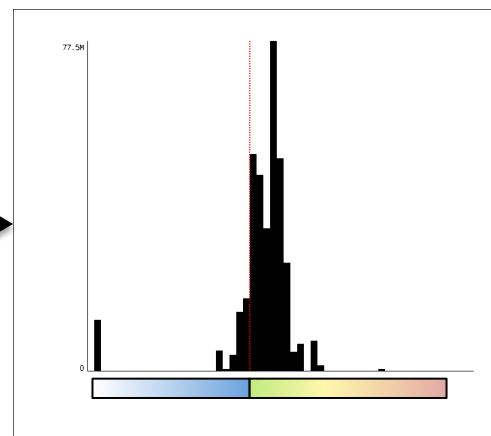
>0.5% - 9:45



>0.1% - 15:45



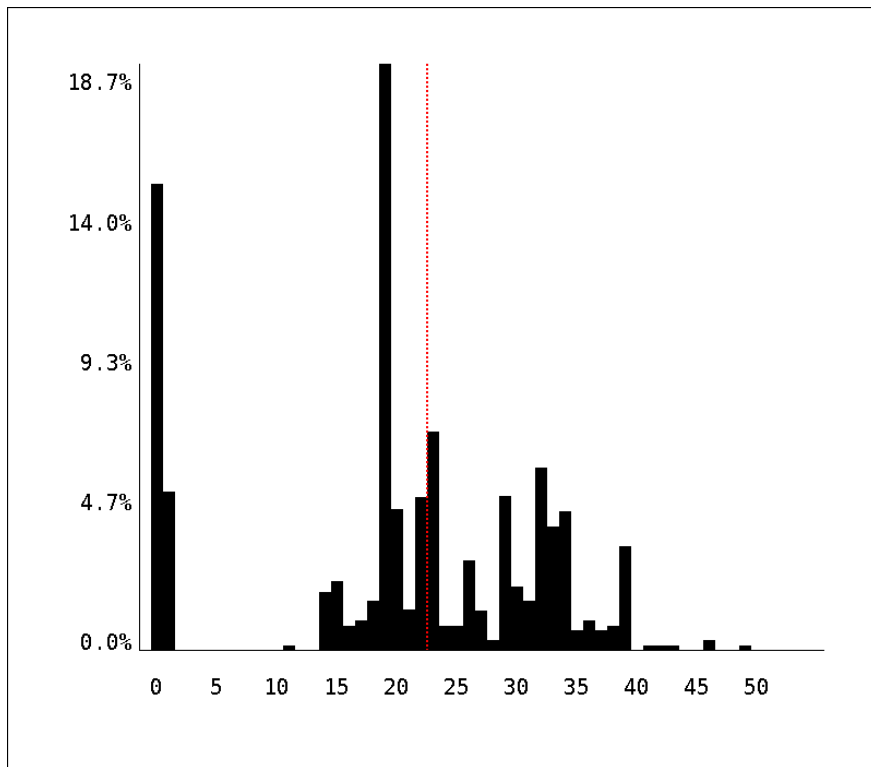
>0.05% - 23:60



Full - 28:71

CLAMR

- Cell-based Adaptive Mesh Refinement
 - Los Alamos National Lab



```
APPLICATION: "clamr_cpuonly" Prec=50
MODULE: 0x400000 "MallocPlus.cpp" Prec=0 [14 instruction(s)]
  FUNC: 0x425508 "Mesh::Mesh" Prec=0 [14 instruction(s)]
MODULE: 0x400000 "clamr_cpuonly.cpp" Prec=1 [17 instruction(s)]
  FUNC: 0x40f9e4 "main" Prec=1 [3 instruction(s)]
  FUNC: 0x40fd2 "do_calc" Prec=1 [14 instruction(s)]
MODULE: 0x400000 "display.c" Prec=0 [16 instruction(s)]
  FUNC: 0x422836 "init_display" Prec=0 [4 instruction(s)]
  FUNC: 0x422abd "draw_scene" Prec=0 [4 instruction(s)]
  FUNC: 0x422c09 "keyPressed" Prec=0 [8 instruction(s)]
MODULE: 0x400000 "hash.c" Prec=0 [73 instruction(s)]
  FUNC: 0x437588 "compact_hash_init" Prec=0 [35 instruction(s)]
  FUNC: 0x438c72 "write_hash_collision_report" Prec=0 [15 instruction(s)]
  FUNC: 0x439d96 "read_hash_collision_report" Prec=0 [15 instruction(s)]
  FUNC: 0x439ed2 "final_hash_collision_report" Prec=0 [8 instruction(s)]
MODULE: 0x400000 "mesh.cpp" Prec=21 [205 instruction(s)]
  FUNC: 0x422d94 "Mesh::write_grid" Prec=21 [14 instruction(s)]
  FUNC: 0x4243e4 "Mesh::print" Prec=21 [2 instruction(s)]
  FUNC: 0x424e32 "Mesh::compare_coordinates_cpu_local_to_cpu_g..." Prec=21 [5 instruction(s)]
  FUNC: 0x427186 "Mesh::refine_smooth" Prec=21 [1 instruction(s)]
  FUNC: 0x428e69 "Mesh::kdtree_setup" Prec=21 [2 instruction(s)]
  FUNC: 0x428f98 "Mesh::calc_spatial_coordinates" Prec=21 [13 instruction(s)]
  FUNC: 0x42944c "Mesh::calc_minmax" Prec=21 [3 instruction(s)]
  FUNC: 0x42964e "Mesh::calc_centerminmax" Prec=21 [6 instruction(s)]
  FUNC: 0x429b62 "Mesh::rezone_all" Prec=21 [58 instruction(s)]
  FUNC: 0x42cc05 "Mesh::calc_neighbors" Prec=21 [37 instruction(s)]
  FUNC: 0x42e754 "Mesh::calc_neighbors_local" Prec=21 [37 instruction(s)]
  FUNC: 0x430d10 "Mesh::print_calc_neighbor_type" Prec=21 [3 instruction(s)]
  FUNC: 0x430f8a "Mesh::calc_symmetry" Prec=21 [24 instruction(s)]
MODULE: 0x400000 "partition.cpp" Prec=0 [50 instruction(s)]
  FUNC: 0x432930 "Mesh::partition_measure" Prec=0 [18 instruction(s)]
  FUNC: 0x433a9a "Mesh::print_partition_measure" Prec=0 [10 instruction(s)]
  FUNC: 0x4341b6 "Mesh::partition_cells" Prec=0 [22 instruction(s)]
MODULE: 0x400000 "state.cpp" Prec=50 [822 instruction(s)]
  FUNC: 0x4123fc "State::add_boundary_cells" Prec=0 [5 instruction(s)]
  FUNC: 0x414ef0 "State::set_timestep" Prec=14 [11 instruction(s)]
  FUNC: 0x415150 "State::fill_circle" Prec=26 [3 instruction(s)]
  FUNC: 0x41574a "State::calc_finite_difference" Prec=50 [541 instruction(s)]
  FUNC: 0x419bb4 "State::symmetry_check" Prec=0 [26 instruction(s)]
  FUNC: 0x41a14a "State::calc_refine_potential" Prec=20 [35 instruction(s)]
  FUNC: 0x41a90e "State::mass_sum" Prec=21 [13 instruction(s)]
  FUNC: 0x41ac7a "State::output_timing_info" Prec=0 [127 instruction(s)]
  FUNC: 0x41c53a "State::parallel_timer_output" Prec=0 [2 instruction(s)]
  FUNC: 0x41c952 "State::compare_state_cpu_local_to_cpu_global..." Prec=0 [9 instruction(s)]
  FUNC: 0x41cd64 "State::print" Prec=0 [3 instruction(s)]
  FUNC: 0x41d28c "State::print_local" Prec=0 [6 instruction(s)]
  FUNC: 0x41deeef "U_halfstep" Prec=39 [21 instruction(s)]
  FUNC: 0x41e0e9 "U_fullstep" Prec=38 [6 instruction(s)]
  FUNC: 0x41e149 "v_corrector" Prec=19 [14 instruction(s)]
```

CLAMR

- Original program
 - Mass preserved to twelve digits
 - Average CPU time: 89.7s
- After CRAFT-assisted conversion
 - Mass preserved to seven digits
 - CPU time speedup (7.3%)
 - Memory usage reduced (~7%)
- After GPU mixed-precision rewrite
 - Up to 4X speed improvement in certain subroutines



Reduced Precision: Conclusions

- Automated analysis can identify fine-grained precision level requirements
- Reduced-precision analysis provides results more quickly than mixed-precision analysis

Future Work

- Short term projects + planned collaborations
 - Visualization/graphics studies (JMU)
 - Development cycle integration
 - Mixed-precision feedback in IDE
 - Correctness/accuracy integration testing
 - Machine learning techniques (UMD)
 - Can we predict low-sensitivity portions of code?
 - Energy-aware analysis (LLNL)
 - Min-maxing energy/performance with a hard bound on accuracy
 - Further case studies (LANL, JMU)



Future Work

- Long term
 - Direct GPU support and/or CUDA implementation
 - Full shadow value analysis
 - Compiler-based implementation
 - Probably LLVM (see Precimonious for similar work)
 - Automatic program transformations
 - Program modeling and verification
 - Guarantees for ALL inputs
 - Can concolic execution help here?
 - Probabilistic arithmetic



“Grand Vision”

- A suite of tools and analysis techniques
 - Understand floating-point behavior and precision-level profiles
 - Recommend or build mixed-precision variants
 - Preserve accuracy
 - Improve performance and reduce energy use
 - Encourage best practices in floating-point code

Contact Info

- Collaborators -

Jeff Hollingsworth (advisor) and Pete Stewart (UMD)

Bronis de Supinski, Matt Legendre, et al. (LLNL)

Bob Robey and Nathan DeBardeleben (LANL)

Email: [lam2mo \(at\) jmu.edu](mailto:lam2mo@jmu.edu)



Personal website: blog.freearrow.com/about

CRAFT website: sf.net/p/crafthpc



Intel
XED2

