HPCToolkit for Top-Down Analyzing Node Performance

Robert Fowler

John Mellor-Crummey Nathan Tallent
Gabriel Marin
(and others)

Department of Computer Science
Rice University

(soon: http://www.cs.rice.edu/~rjf/newstuff/RiceTools_5_2003.ppt)

Outline

- Other Stuff
  - Motivation
  - Compilation Tools
- HPCTools Toolkit
  - hpcview
  - bloop
  - what you can do with it

Background

What we (Rice Parallel Compilers Group) do –

- Code optimization
  - Aggressive (mostly source-to-source) compilation.
  - Hand application of optimizing transformations.
    - Try out transformations by hand first.
    - Work on real codes with algorithm, library, and application developers.

We spend a lot of time (too much?) analyzing executions. Why?

1. Deeply pipelined, out of order, superscalar processors with non-blocking caches and deep memory hierarchies.
2. Aggressive (-O4), bad, and/or idiosyncratic vendor compilers.
3. Amdahl's law says that you have to get all the pieces right.

What we did --

- Built tools to meet our needs w.r.t. the run/analyze/tune cycle.
- They were so useful that we are now distributing them

Why I'm here --

- To kickstart some collaboration on performance work.
- To present the tools
- To identify targets of opportunity for future projects.

Short to long term interests

Immediate Impact Supporting Applications in "Expeditions"

Performance diagnosis tools.

Hand application of aggressive transformations to important codes.

Source-to-source transformation tools.

Scalable, rapidly converging algorithms for applications

High Perf. Libraries and Components

Compiler optimization for Explicit SPMD Languages (UPC, CAF)

Fundamental compiler algorithms.

High Performance Script-Programming Systems

Compilation for Processor-in-Memory Architectures.

Long Term Research Affecting Future HPC Systems

Work towards vertical integration along this spectrum.

Itanium-2 Compiler/Performance Fun!

(900MHz dual-processor McKinley)

(1)

NAS SP	Hand-coded	dHPF
ORC 2.0	325 sec	345 sec
EFC 7.1	175 sec	610 sec

(2) WRF sequential execution times:

900 MHz I2 → .67 sec/iteration

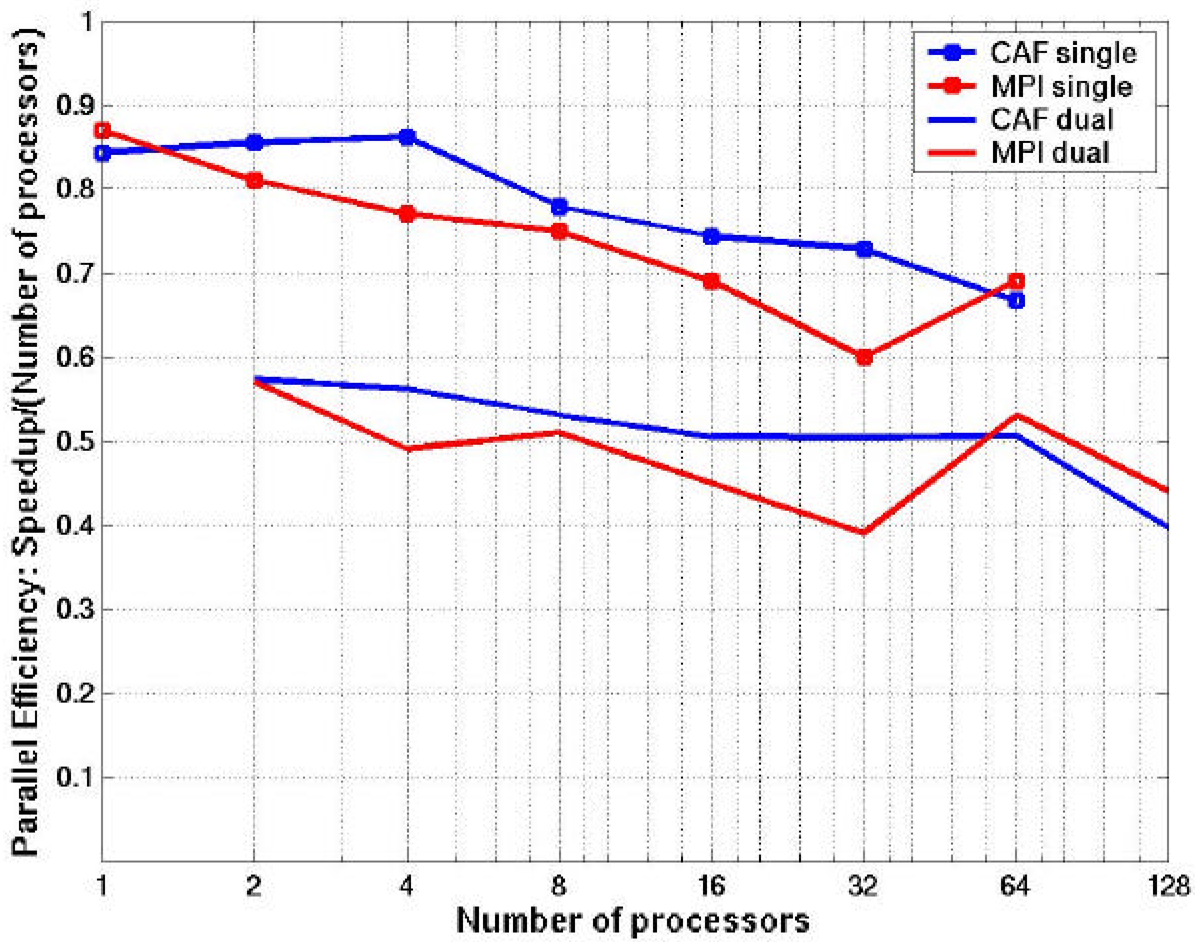
667 MHz EV67 → .92 sec/iteration

(3) efc does not have a “-g3”option, i.e. you can either Profile and debug or turn on optimization.

The Rice Co-Array Fortran Compiler

- Near production-quality F90 front end from Open64
- Working prototype for a CAF subset
  - allocate co-arrays using static constructor-like strategy
  - co-array access
    - remote data access uses ARMCI get/put
    - process local data access uses load/store
  - synch_all, synch_team synchronization
  - multi-dimensional array section operations
- Successfully compiled and executed NAS MG
  - platforms: SGI Origin, Itanium2 + Myrinet
  - performance similar to hand-coded MPI
- We haven't started to apply aggressive optimizations.

NAS MG Efficiency (Class C)



Itanium-2/
Myrinet 2000/
efc

Source-to-source loop translator

Current capabilities

- Apply code motion to enable fusion
- Skew spatial loops to enable fusion and blocking
- Fuse loops at multiple levels
- Reduce storage for fused and blocked code
- Block and unroll loops to reuse values in registers
- Generate code with a guard-free core

Planned enhancements

- Skew temporal loops to improve temporal locality

Status

- Ready for user trials if you can run on SPARC or SGI
- Porting (~.5M lines of legacy C++) to Linux/IA32 compilers.

Performance Results

Automatic N-level fusion and storage optimization applied to the NCOMMAS Runga-Kutta Kernel. (Origin 2k)

Code	Cycles	L1 Misses	L2 Misses	TLB Misses
Original	1	1	1	1
1-level fusion	0.94	1.31	0.85	0.45
1-level fusion w/ array contraction	0.67	1.42	0.41	0.06
2-level fusion w/ array contraction	0.69	1.14	0.39	0.06
1-level fusion w/ unroll-and-jam	0.94	1.16	0.72	0.47

Hardware Performance Counters

- What they do
  - count "events"
  - record information about an instruction as it executes
- Utility
  - capture information about performance critical details that is otherwise inaccessible
    - cache and TLB misses, mispredicted branches, etc.
- Ways to exploit them
  - instrument code to start, stop, read, reset counters
    - typically a manual process
  - sample events during execution

Sample-based Performance Analysis

- Each time an event threshold count is exceeded
  - —sample the program counter
  - —record it in a histogram
- Map sampled PC values back to source lines
- Advantages
  - —provides a high-level view of where "events" happen during execution
  - —can be started at launch time without prior preparation

NOTE: on Alpha EV67, most sampling is instruction based rather than event based

hpcview

A tool for exploring profile-like data

- Evaluate node performance of large scientific applications
- Pinpoint key code fragments and issues
- Complementary to tools for analyzing parallel efficiency

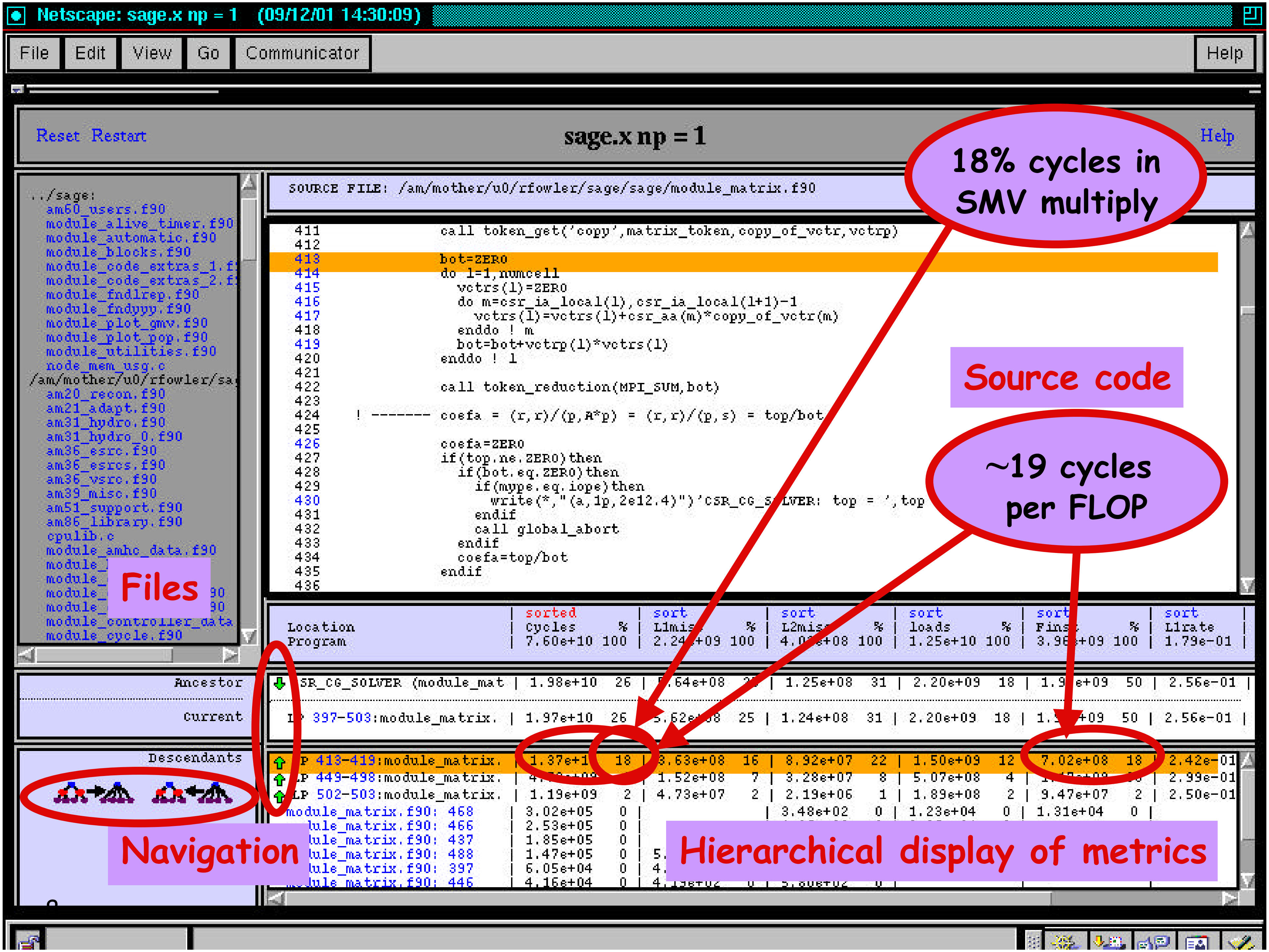
Approach

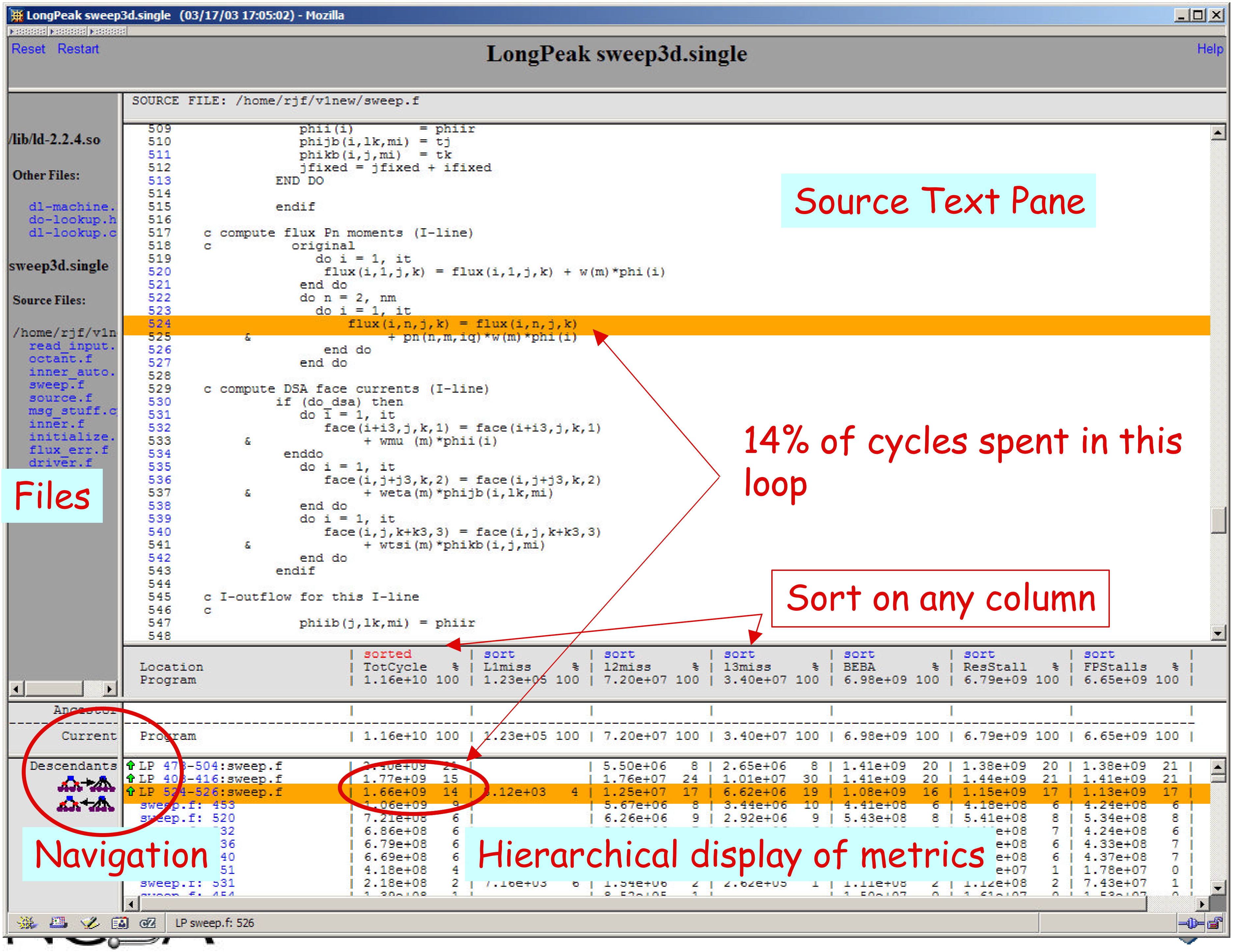
- **Gather multiple performance metrics from hardware performance counters, simulation, and other sources**
  - ssrun on SGI, papirun on Linux, DCPI and uprofile on Tru64.
- **Compute user-specified derived tuning metrics with an expression interpreter**
  - e.g. wasted time = cycles - FLOPS
- **Correlate metrics with program structure and source code**
- **Provide integrated user interface based on hypertext browsing technology**

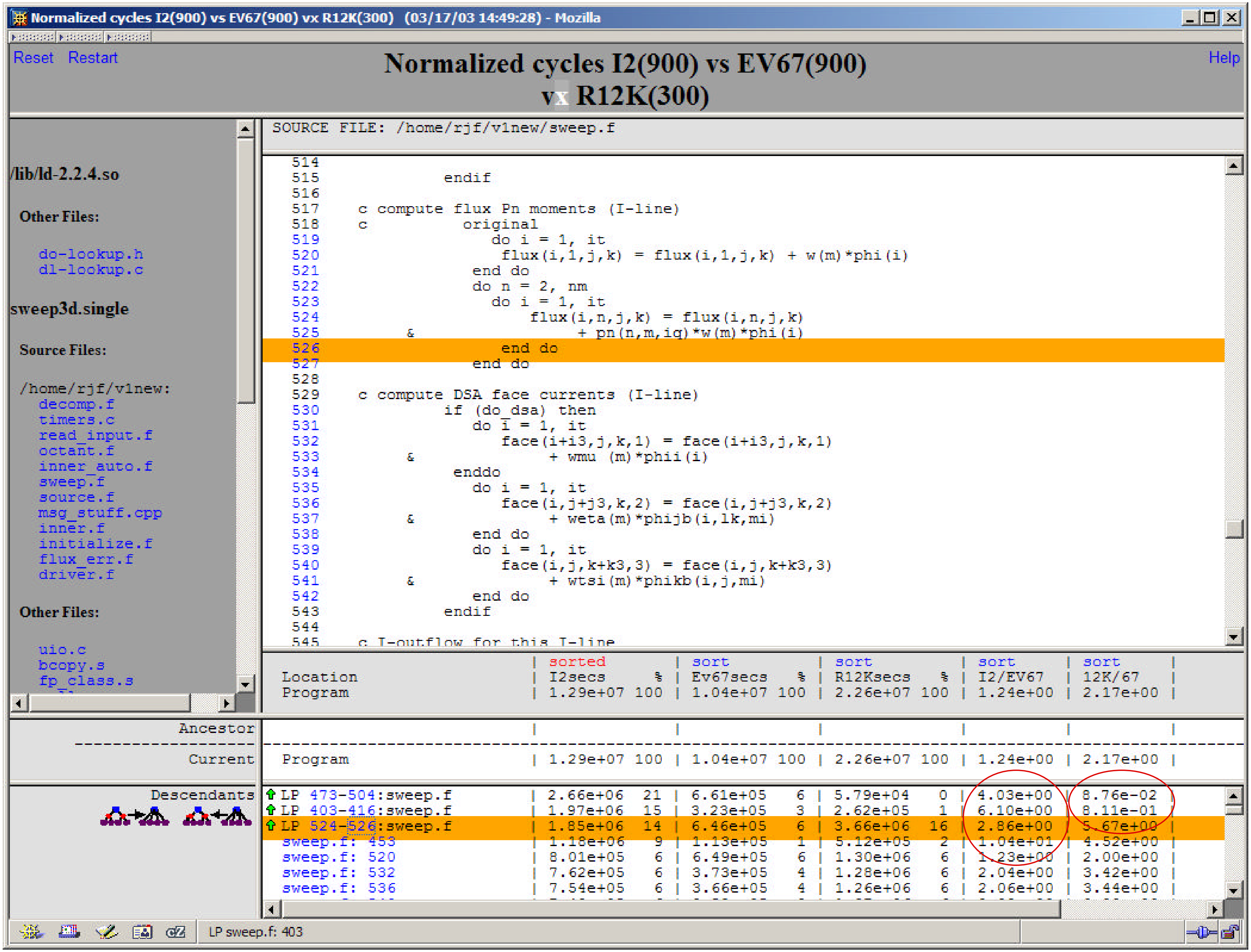
Works for all languages, programming models and large applications

Support a Hierarchy of Views

- **Problem**
  - line-level performance statistics may be inaccurate, and offer a myopic view of program performance
- **Goal**
  - language-independent solution that enables construction of hierarchical program views
- **Approach**
  - recover program structure through analysis of application binaries with bloop tool


Netscape: sage.x np = 1 (09/12/01 14:30:09)
File
Edit
View
Go
Communicator
Help
Reset Restart
sage.x np = 1
Help
18% cycles in SMV multiply
SOURCE FILE: /am/mother/u0/rfowler/sage/sage/module_matrix.f90
../sage:
am60_users.f90
module_alive_timer.f90
module_automatic.f90
module_blocks.f90
module_fndlrep.f90
module_fndyyy.f90
module_plot_gmv.f90
module_plot_pop.f90
module_utilities.f90
node_mem_usg.c
am20_recon.f90
am21_adapt.f90
am31_hydro.f90
am31_hydro_0.f90
am36_esrc.f90
am36_esrcs.f90
am36_vsrc.f90
am39_misc.f90
am51_support.f90
am86_library.f90
cpulib.c
module_amhc_data.f90
module_cycle.f90
Files
411 call token_get('copy',matrix_token,copy_of_vctr,vctrp)
413 bot=ZERO
414 do l=1,numcell
415 vctrs(l)=ZERO
416 do m=csr_ia_local(l),csr_ia_local(l+1)-1
417 vctrs(l)=vctrs(l)+csr_aa(m)*copy_of_vctr(m)
418 enddo ! m
419 bot=bot+vctrp(l)*vctrs(l)
420 enddo ! l
422 call token_reduction(MPI_SUM,bot)
424 ! ------- coefa = (r,r)/(p,A*p) = (r,r)/(p,s) = top/bot
426 coefa=ZERO
427 if(top.ne.ZERO)then
428 if(bot.eq.ZERO)then
429 if(mype.eq.iope)then
431 endif
432 call global_abort
433 endif
434 coefa=top/bot
435 endif
Source code
~19 cycles per FLOP
Location
Program
sorted cycles % 7.60e+10 100
sort loads % 1.25e+10 100
sort L1rate 1.79e-01
Ancestor
Current
Descendants
397-503:module_matrix. 1.97e+10 26
413-419:module_matrix. 1.37e+1 18
449-498:module_matrix.
502-503:module_matrix. 1.19e+09 2 4.73e+07 2 2.19e+06 1 1.89e+08 2 9.47e+07 2 2.50e-01
Navigation
Hierarchical display of metrics


LongPeak sweep3d.single (03/17/03 17:05:02) - Mozilla
Reset Restart
LongPeak sweep3d.single
Help
SOURCE FILE: /home/rjf/v1new/sweep.f
Source Text Pane
14% of cycles spent in this loop
Sort on any column
Files
Navigation
Hierarchical display of metrics
/lib/ld-2.2.4.so
Other Files:
sweep3d.single
Source Files:
Location
Program
sorted TotCycle % 1.16e+10 100
sort L1miss % 1.23e+05 100
sort l2miss % 7.20e+07 100
sort l3miss % 3.40e+07 100
sort BEBA % 6.98e+09 100
sort ResStall % 6.79e+09 100
sort FPStalls % 6.65e+09 100
Current
Descendants
LP sweep.f: 526


Normalized cycles I2(900) vs EV67(900) vx R12K(300) (03/17/03 14:49:28) - Mozilla
Reset Restart
Normalized cycles I2(900) vs EV67(900) vx R12K(300)
Help
/lib/ld-2.2.4.so
Other Files:
do-lookup.h
dl-lookup.c
sweep3d.single
Source Files:
/home/rjf/v1new:
decomp.f
timers.c
read_input.f
octant.f
inner_auto.f
sweep.f
source.f
msg_stuff.cpp
inner.f
initialize.f
flux_err.f
driver.f
Other Files:
uio.c
bcopy.s
fp_class.s
SOURCE FILE: /home/rjf/v1new/sweep.f
514
515 endif
516
517 c compute flux Pn moments (I-line)
518 c original
519 do i = 1, it
520 flux(i,1,j,k) = flux(i,1,j,k) + w(m)*phi(i)
521 end do
522 do n = 2, nm
523 do i = 1, it
524 flux(i,n,j,k) = flux(i,n,j,k)
525 & + pn(n,m,iq)*w(m)*phi(i)
526 end do
527 end do
528
529 c compute DSA face currents (I-line)
530 if (do_dsa) then
531 do i = 1, it
532 face(i+i3,j,k,1) = face(i+i3,j,k,1)
533 & + wmu (m)*phii(i)
534 enddo
535 do i = 1, it
536 face(i,j+j3,k,2) = face(i,j+j3,k,2)
537 & + weta(m)*phijb(i,lk,mi)
538 end do
539 do i = 1, it
540 face(i,j,k+k3,3) = face(i,j,k+k3,3)
541 & + wtsi(m)*phikb(i,j,mi)
542 end do
543 endif
544
545 c I-outflow for this I-line
Location | sorted I2secs % | sort Ev67secs % | sort R12Ksecs % | sort I2/EV67 | sort 12K/67 |
Program | 1.29e+07 100 | 1.04e+07 100 | 2.26e+07 100 | 1.24e+00 | 2.17e+00 |
Ancestor
Current Program | 1.29e+07 100 | 1.04e+07 100 | 2.26e+07 100 | 1.24e+00 | 2.17e+00 |
Descendants
LP 473-504:sweep.f | 2.66e+06 21 | 6.61e+05 6 | 5.79e+04 0 | 4.03e+00 | 8.76e-02 |
LP 403-416:sweep.f | 1.97e+06 15 | 3.23e+05 3 | 2.62e+05 1 | 6.10e+00 | 8.11e-01 |
LP 524-526:sweep.f | 1.85e+06 14 | 6.46e+05 6 | 3.66e+06 16 | 2.86e+00 | 5.67e+00 |
sweep.f: 453 | 1.18e+06 9 | 1.13e+05 1 | 5.12e+05 2 | 1.04e+01 | 4.52e+00 |
sweep.f: 520 | 8.01e+05 6 | 6.49e+05 6 | 1.30e+06 6 | 1.23e+00 | 2.00e+00 |
sweep.f: 532 | 7.62e+05 6 | 3.73e+05 4 | 1.28e+06 6 | 2.04e+00 | 3.42e+00 |
sweep.f: 536 | 7.54e+05 6 | 3.66e+05 4 | 1.26e+06 6 | 2.06e+00 | 3.44e+00 |
LP sweep.f: 403

Assessment of HPCView Functionality

- Top down analysis focuses attention where it belongs
  - —sorted views put the important things first
- Integrated browsing interface facilitates exploration
  - —rich network of connections makes navigation simple
- Hierarchical, loop-level reporting facilitates analysis
  - —more sensible view when statement-level data is imprecise
- Binary analysis handles multi-lingual applications and libraries
  - —succeeds where language and compiler based tools can't
- Sample-based profiling, aggregation and derived metrics
  - —reduce manual effort in analysis and tuning cycle
- Multiple metrics provide a better picture of performance
- Platform independent analysis tool
- Limited by quality of the compiler, instrumentation.

bloop

- Recovers loop nesting structure from application binaries
  - —identify basic blocks
  - —recover control flow graph (CFG)
  - —identify natural loop nests in CFG
- Map machine instructions to source lines using symbol table
- Normalize output to recover source-level view
- Current binary formats
  - —MIPS, Alpha, Pentium4, IA64, Sparc

Recovered Program Structure

```
<PGM n="/apps/smg98/test/smg98">
 ...
 <F n="/apps/smg98/struct_linear_solvers/smg_relax.c">
   <P n="hypre_SMGRelaxFreeARem">
     <L b="146" e="146">
       <S b="146" e="146"/>
     </L>
   </P>
   <P n="hypre_SMGRelax">
     <L b="297" e="328">
       <S b="297" e="297"/>
       <L b="301" e="328">
         <S b="301" e="301"/>
         <L b="318" e="325">
           <S b="318" e="325"/>
         </L>
         <S b="328" e="328"/>
       </L>
       <S b="302" e="302"/>
     </L>
   </P>
   ...
 </F>
</PGM>
```

Program

File

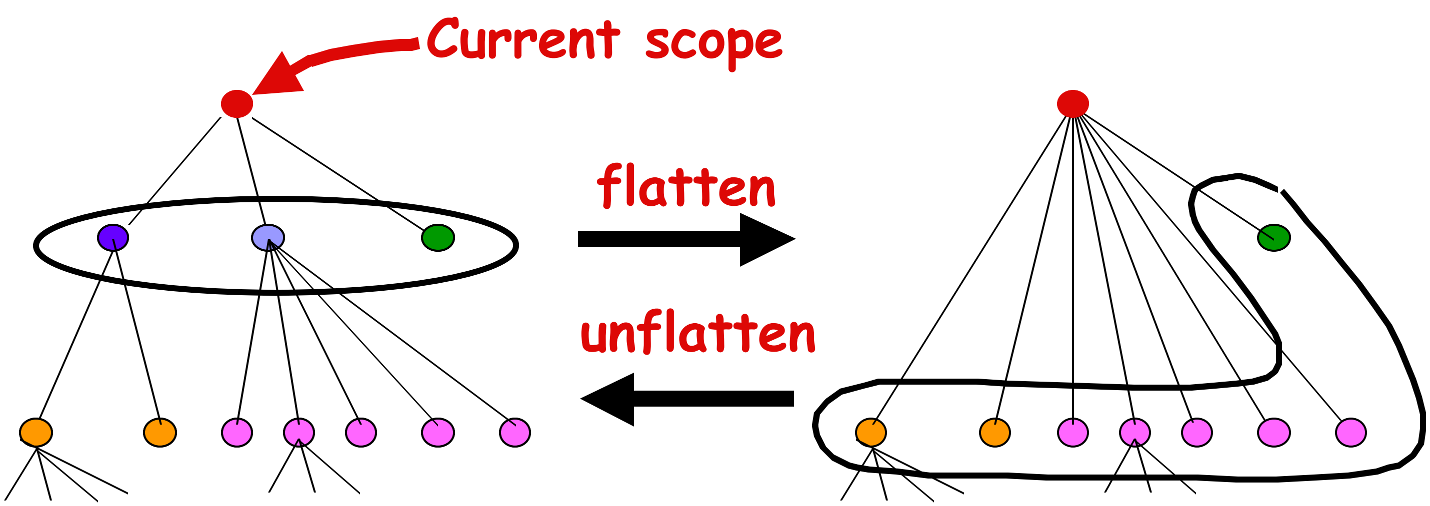
Procedure

Loop

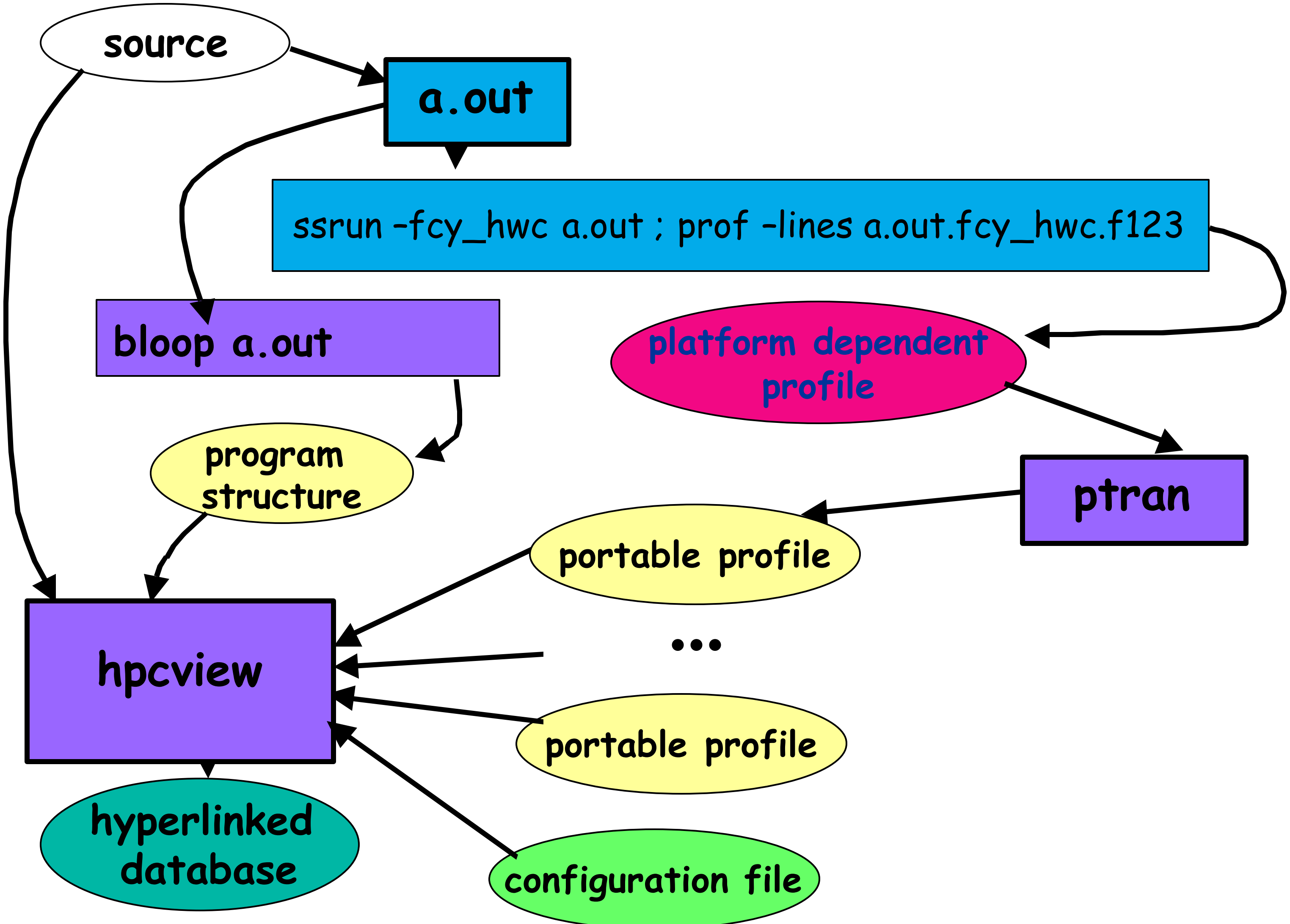
Statement

Flattening for Top Down Analysis

- Problem
  - —strict hierarchical view of a program is too rigid
  - —want to compare program components at the same level as peers
- Solution
  - —enable a scope's descendants to be flattened to compare their children as peers



Using HPCTools Toolkit



hpcview Configuration File

```
<HPCVIEW>
  <TITLE name="POP 4-way shmem, model size=medium" />
<PATH name="." />
<PATH name="./compile" />
<PATH name="../sshmem" />
<PATH name="../source" /> -
<METRIC name="pcc" displayName="Cycles">
     <FILE name="pop.fcy_hwc.pxml" />
</METRIC>
<METRIC name="dc" displayName="L1 miss">
     <FILE name="pop.fdc_hwc.pxml" />
</METRIC>
<METRIC name="dsc" displayName="L2 miss">
     <FILE name="pop.fdsc_hwc.pxml" />
</METRIC>
<METRIC name="fp" displayName="FP insts">
     <FILE name="pop.fgfp_hwc.pxml" />
</METRIC>
<METRIC name="rat" displayName="cy per FLOP">-
     <COMPUTE>
         <math> <apply> <divide/> <ci>pcc</ci> <ci>fp</ci> </apply> </math>
     </COMPUTE>
</METRIC>
</HPCVIEW>
```

Heading on Display

Paths to interesting source directories

Metrics defined by Platform Independent Profile Files

Expression for derived metric

Some Uses for hpcview

- Identifying unproductive work
  - —where is the program spending its time not performing FLOPS
- Memory hierarchy utilization
  - —bandwidth utilization: misses x line size/cycles
  - —is prefetching being used? is it helping or hurting?
- Cross architecture comparisons
  - —what program features cause performance differences?
- Explore the gap between peak and observed performance
  - —(max instructions per cycle * cycles) - graduated instructions
- Evaluating load balance in a parallelized code
  - —how do profiles for different processes compare

papirun/papiprof

- papirun: open-source equivalent of SGI's "ssrun"
  - initiate sample-based profiling of an execution
  - collect data about user program and all dynamic libraries
  - current implementation for Linux
- papiprof: prof-like tool for use with papirun
  - interpret profiles collected with papirun
  - output styles
    - ascii profile format
    - XML-based profile format for use with HPCView

Getting/Using HPCToolkit

At NCSA – Currently no supported installations.

(Will work anywhere PAPI can generate profiles.)

Old URL: http://www.cs.rice.edu/~dsystem/hpctools/

New URL: http://hipersoft.cs.rice.edu/hpctoolkit

Getting Started:

Old distribution regime -- Source and a variety of binary tarballs.

New distribution regime -- Use CVS to get a source distribution.

Using an installed copy:

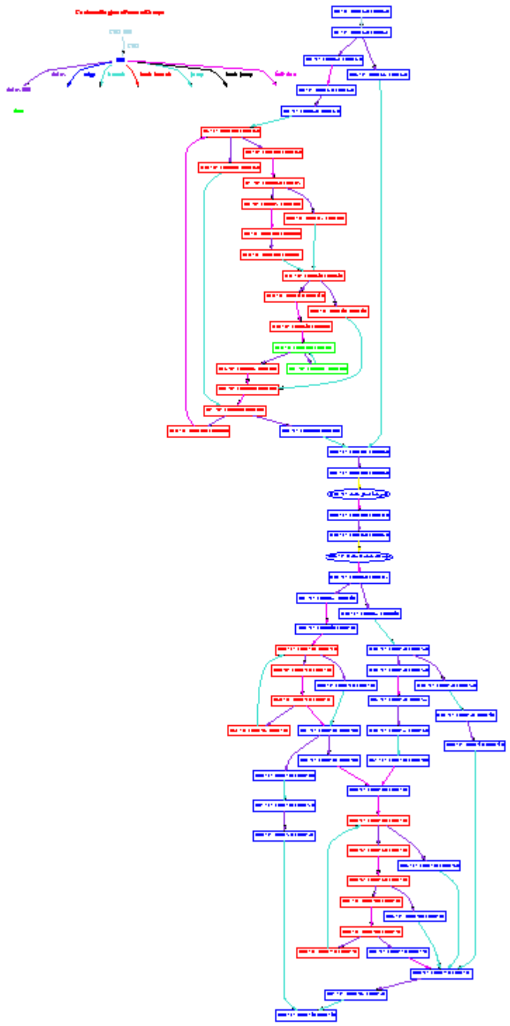
setenv HPCTOOLKIT <where it's installed>

source $HPCTOOLKIT/Sourceme-csh

(Modify examples in the distribution.)

The End

Sample Flowgraph from an Executable



Loop nesting structure

- blue: outermost level
- red: loop level 1
- green loop level 2

Observation
optimization complicates program structure!

Normalizing Program Structure

Constraint: each source line must appear at most once

Coalesce duplicate lines

(1) if duplicate lines appear in different loops

- find least common ancestor in scope tree; merge corresponding loops along the paths to each of the duplicates

    purpose: re-rolls loops that have been split

(2) if duplicate lines appear at multiple levels in a loop nest

- discard all but the innermost instance

    purpose: handles loop-invariant code motion

apply (1) and (2) repeatedly until a fixed point is reached