



Toward portable programming of numerical linear algebra on manycore nodes

Michael A. Heroux
Scalable Algorithms Department
Sandia National Laboratories

Collaborators:

SNL Staff: [B.|R.] Barrett, E. Boman, R. Brightwell, H.C. Edwards, A. Williams

SNL Postdocs: M. Hoemmen, S. Rajamanickam, M. Wolf

ORNL staff: Chris Baker



Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have `MPI_Init()`.
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
5. DRY is not possible across CPUs and GPUs.
6. Extended precision is too expensive to be useful.
7. Resilience will be built into algorithms.
8. GPUs are a harbinger of CPU things to come.
9. Fortran Developers are in trouble in a manycore world.
10. Global SIMT is sufficient parallelism for scientific computing.

Trilinos Contributors

Current Contributors

Chris Baker

Ross Bartlett
Pavel Bochev
Erik Boman
Lee Buermann
Todd Coffey
Eric Cyr
David Day
Karen Devine
Clark Dohrmann
David Gay
Glen Hansen
David Hensinger
Mike Heroux
Mark Hoemmen
Russell Hooper
Jonathan Hu
Sarah Knepper
Patrick Knupp
Joe Kotulski
Jason Kraftcheck

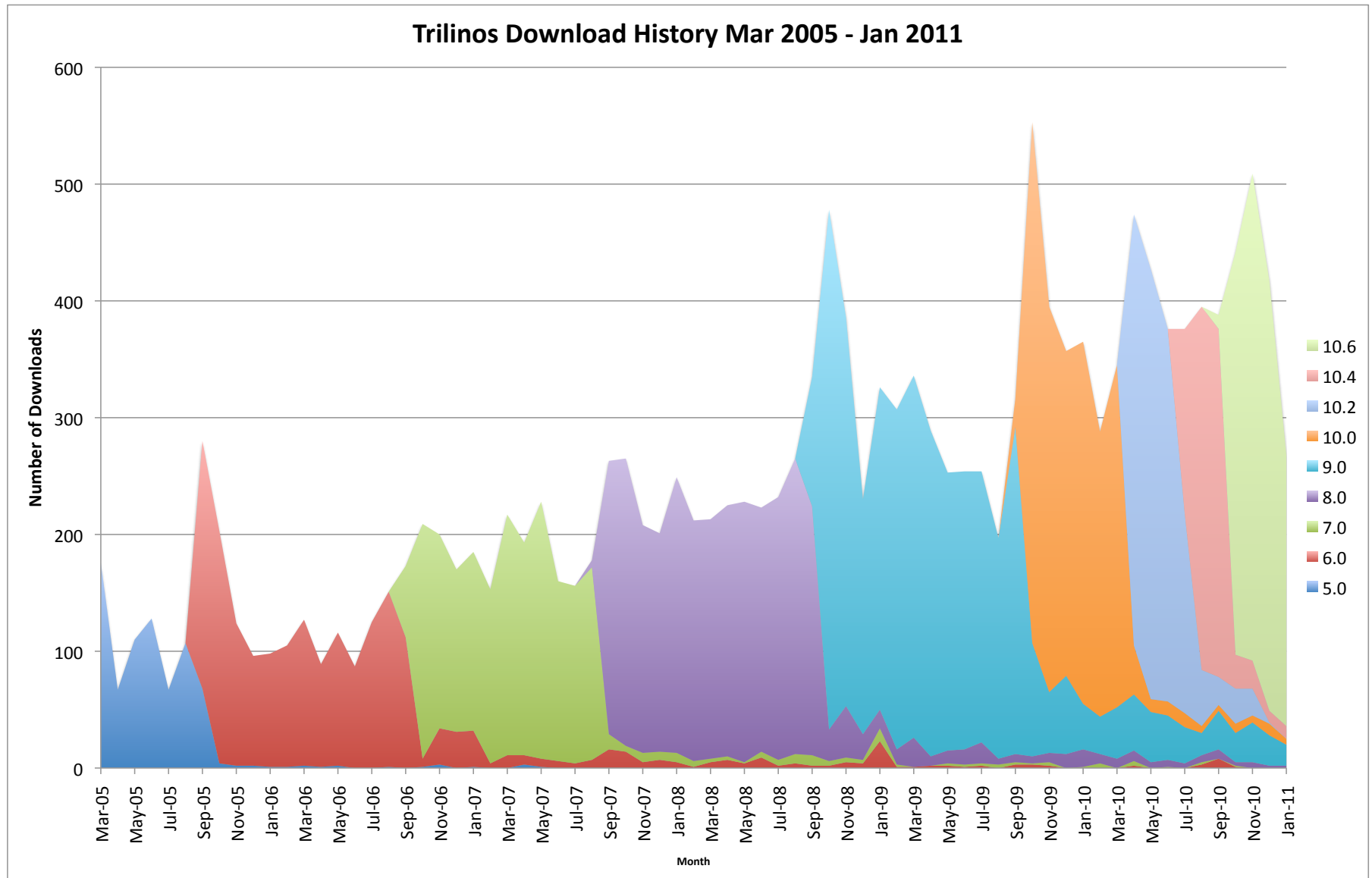
Rich Lehoucq
Nicole Lemaster
Kevin Long
Karla Morris
Chris Newman
Kurtis Nusbaum
Ron Oldfield
Mike Parks
Roger Pawlowski
Brent Perschbacher
Kara Peterson
Eric Phipps
Siva Rajamanickam
Denis Ridzal
Lee Ann Riesen
Damian Rouson
Andrew Salinger
Nico Schlömer
Chris Siefert
Greg Sjaardema
Bill Spatz
Heidi Thornquist
Ray Tuminaro

Jim Willenbring
Alan Williams
Michael Wolf

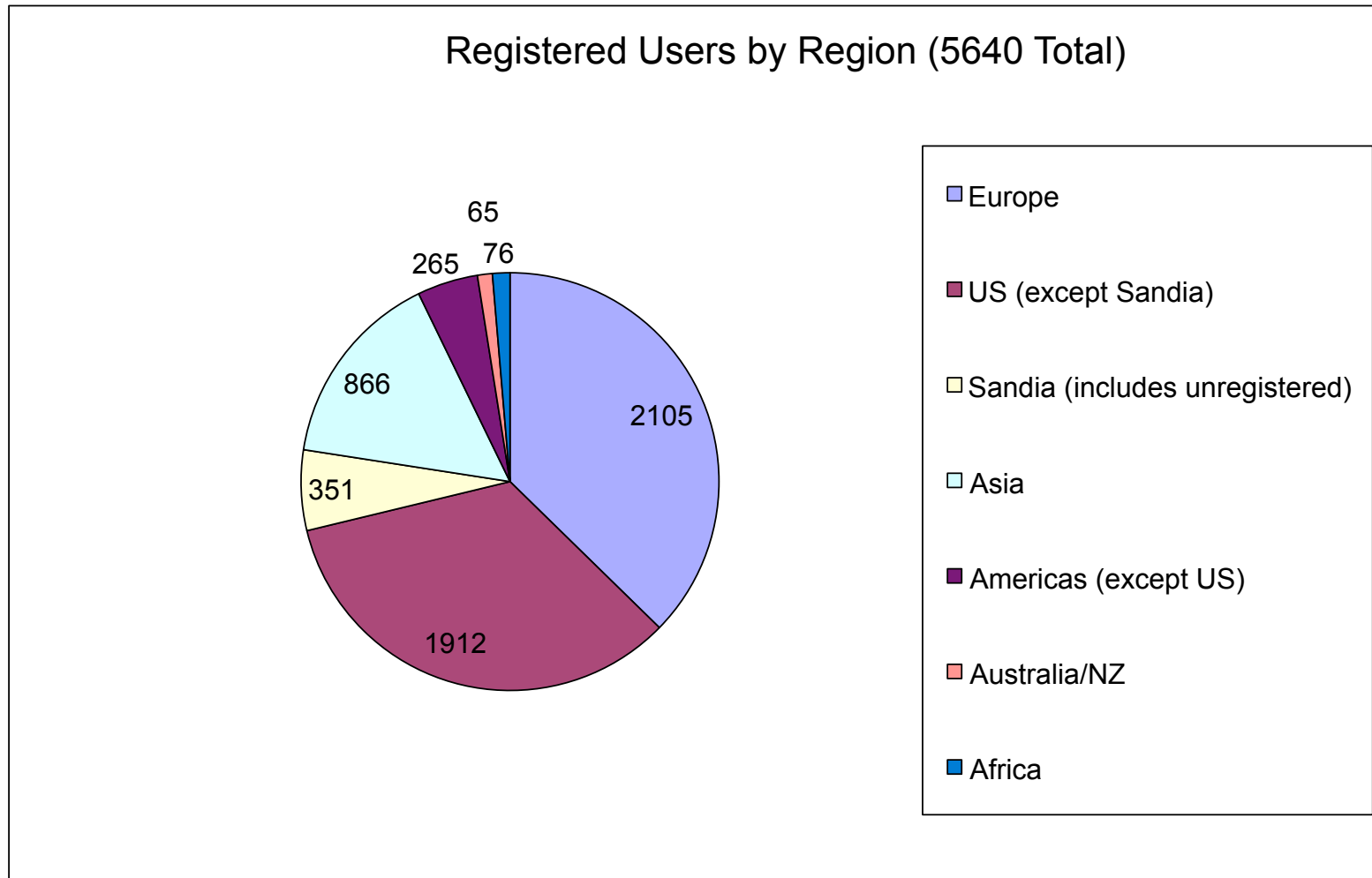
Past Contributors

Paul Boggs
Jason Cross
Michael Gee
Esteban Guillen
Bob Heaphy
Ulrich Hetmaniuk
Robert Hoekstra
Vicki Howle
Kris Kampshoff
Tammy Kolda
Joe Outzen
Mike Phenow
Paul Sexton
Ken Stanley
Marzio Sala
Cedric Chevalier

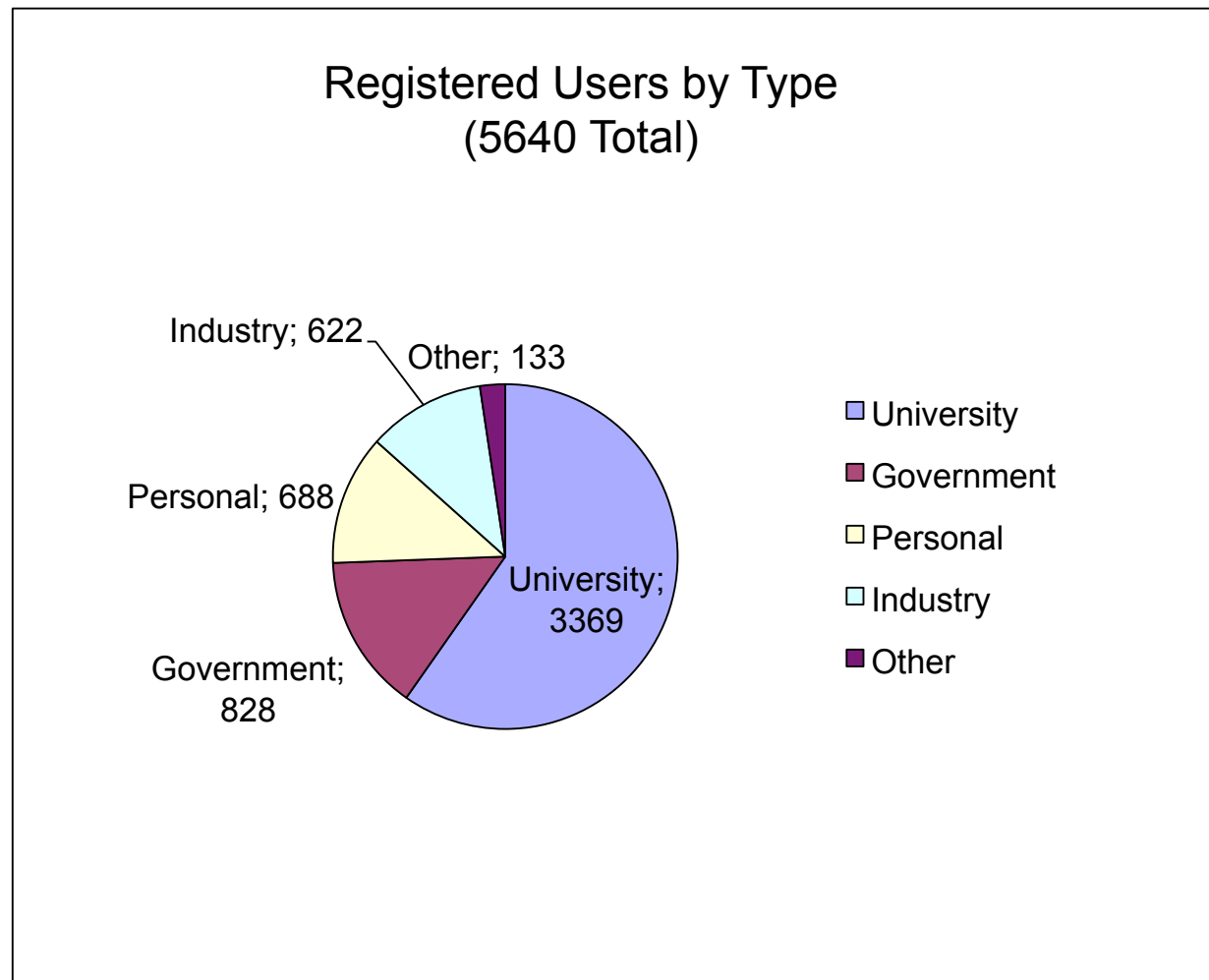
Trilinos Download History: 17600 Total



Registered User by Region



Registered Users by Type



Ubuntu/Debian: Other sources

Ubuntu -- Details of source package trilinos in maverick

http://packages.ubuntu.com/source/maverick/math/trilinos

Search source package names

TrilinosHan...Google Code Report Time Dex White Pa...le for Free SRN Browser Remote.Sandia.Gov Google Maps News (455) Popular How to Conn...sions (RDP)

ubuntu

>> Ubuntu >> Packages >> maverick >> Source >> math >> trilinos

[karmic] [lucid] [maverick]

Source Package: trilinos (10.0.4.dfsg-1.1) [universe]

The following binary packages are built from this source package:

[libtrilinos](#)
parallel solver libraries within an obj

[libtrilinos-dbg](#)
parallel solver libraries within an obj

[libtrilinos-dev](#)
parallel solver libraries within an obj

[libtrilinos-doc](#)
parallel solver libraries within an obj

[python-pytrilinos](#)
parallel solver libraries within an obj

Other Packages Related

- build-depends
- build-depends-ind
- cdbs
common build system for Debi
- quilt
Tool to work with series of pat
- debhelper (>= 7)

Debian -- Details of source package trilinos in sid

http://packages.debian.org/source/sid/trilinos

Search source package names

TrilinosHan...Google Code Report Time Dex White Pa...le for Free SRN Browser Remote.Sandia.Gov Google Maps News (455) Popular How to Conn...sions (RDP)

debian

>> Debian >> Packages >> sid (unstable) >> Source >> math >> trilinos

[squeeze] [sid]

Source Package: trilinos (10.4.0.dfsg-1)

The following binary packages are built from this source package:

[libtrilinos](#)
parallel solver libraries within an object-oriented software framework

[libtrilinos-dbg](#)
parallel solver libraries within an object-oriented software framework

[libtrilinos-dev](#)
parallel solver libraries within an object-oriented software framework

Links for trilinos

Debian Resources:

- [Bug Reports](#)
- [Developer Information \(PTS\)](#)

```
maherou@jaguar13:/ccs/home/maherou> module avail trilinos
```

```
----- /opt/cray/modulefiles -----
```

```
trilinos/10.0.1(default) trilinos/10.2.0
```

```
----- /sw/xt5/modulefiles -----
```

```
trilinos/10.0.4 trilinos/10.2.2 trilinos/10.4.0 trilinos/8.0.3 trilinos/9.0.2
```

[python-central](#)
register and build utility for Py

Download trilinos

- [libopenmpi-dev](#)
high performance message passing library -- header files
- [libsuperlu3-dev](#)
Direct solution of large, sparse systems of linear equations

Capability Leaders: Layer of Proactive Leadership

- Areas:
 - ◆ Framework, Tools & Interfaces (J. Willenbring).
 - ◆ Software Engineering Technologies and Integration (R. Bartlett).
 - ◆ Discretizations (P. Bochev).
 - ◆ Geometry, Meshing & Load Balancing (K. Devine).
 - ◆ Scalable Linear Algebra (M. Heroux).
 - ◆ Linear & Eigen Solvers (J. Hu).
 - ◆ Nonlinear, Transient & Optimization Solvers (A. Salinger).
 - ◆ **Scalable I/O: (R. Oldfield)**
- Each leader provides strategic direction across all Trilinos packages within area.

Trilinos Package Summary

	Objective	Package(s)
Discretizations	Meshing & Discretizations	STKMesh, Intrepid, Pamgen, Sundance, ITAPS, Mesquite
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Services	Linear algebra objects	Epetra, Jpetra, Tpetra, Kokkos
	Interfaces	Thyra, Stratimikos, RTOp, FEI, Shards
	Load Balancing	Zoltan, Isorropia
	“Skins”	PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika
	C++ utilities, I/O, thread API	Teuchos, EpetraExt, Kokkos , Triutils, ThreadPool, Phalanx
Solvers	Iterative linear solvers	AztecOO, Belos, Komplex
	Direct sparse linear solvers	Amesos, Amesos2
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi, Rbgen
	ILU-type preconditioners	AztecOO, IFPACK, Ifpack2
	Multilevel preconditioners	ML, CLAPS
	Block preconditioners	Meros, Teko
	Nonlinear system solvers	NOX, LOCA
	Optimization (SAND)	MOOCHO, Aristos, TriKota, Globipack, Optipack
	Stochastic PDEs	Stokhos



Three Design Points

- Terascale Laptop: Uninode-Manycore
- Petascale Deskside: Multinode-Manycore
- Exascale Center: Manynode-Manycore

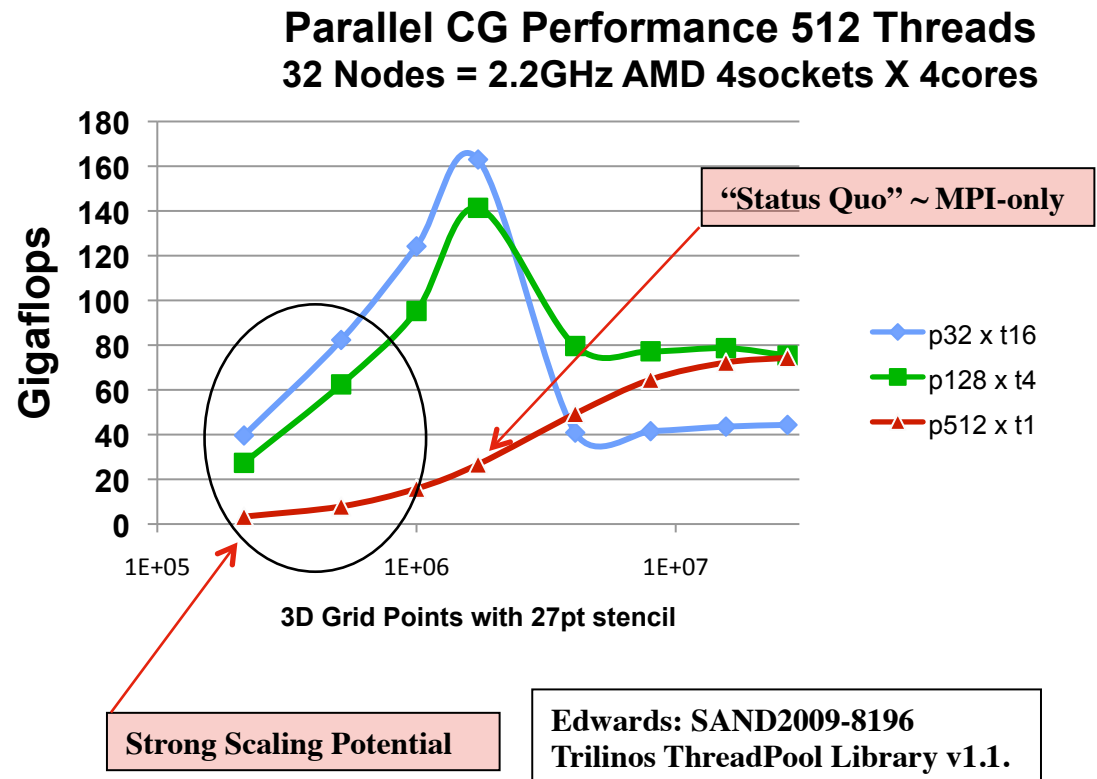


Basic Concerns: Trends, Manycore

- Stein's Law: *If a trend cannot continue, it will stop.*

Herbert Stein, chairman of the Council of Economic Advisers under Nixon and Ford.

- Trends at risk:
 - Power.
 - Single core performance.
 - Node count.
 - Memory size & BW.
 - Concurrency expression in existing Programming Models.
 - Resilience.





Observations

- MPI-Only is not sufficient, except ... much of the time.
- Near-to-medium term:
 - MPI+[OMP|TBB|Pthreads|CUDA|OCL|MPI]
 - Long term, too?
- Concern:
 - Best hybrid performance: 1 MPI rank per UMA core set.
 - UMA core set size growing slowly → Lots of MPI tasks.
- Long- term:
 - Something hierarchical, global in scope.
- Conjecture:
 - Data-intensive apps need non-SPDM model.
 - Will develop new programming model/env.
 - Rest of apps will adopt over time.
 - Time span: 10-20 years.



What Can we Do Right Now?

- Study why MPI was successful.
- Study new parallel landscape.
- Try to cultivate an approach similar to MPI (and others).



MPI Impressions

MPI: It Hurts So Good

Dan Reed, Microsoft

Workshop on the Road Map for the
Revitalization of High End
Computing
June 16-18, 2003

• Observations

- “assembly language” of parallel programming
- lowest common denominator
 - portable across architectures
 - system independent
 - even
- upfront effort required

• C

So What Would Life Be Like Without MPI?

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Serial C

```
long fib_serial(long n)
{
    if (n < 2) return n;
    return fib_serial(n-1) + fib_serial(n-2);
}
```

OpenMP 3.0

```
long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    #pragma omp task default(none) shared(x, n)
    {
        x = fib_parallel(n-1);
    }
    y = fib_parallel(n-2);
    #pragma omp taskwait
    return (x+y);
}
```

Cilk++

```
long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    x = cilk_spawn fib_parallel(n-1);
    y = fib_parallel(n-2);
    cilk_sync;
    return (x+y);
}
```

Tim Stitts, CSCS
SOS14 Talk
March 2010

Fortran

```
def fib_ser
{
    var x,y; n:1
    if (n < 2) then
        cobegin {
            x=fib_ser(n-1);
            y=fib_ser(n-2);
        }
    return x+y;
}
```

MPI

```
fib_parallel(n: ZZ): ZZ requir
if n < 2 then n else fib_pa
```



“ MPI is often considered the
“portable assembly language” of
parallel computing, ...”

Brad Chamberlain, Cray, 2000.



3D Stencil in NAS MG



```
subroutine comm(n1,n2,n3,kk)
  use caf_intrinsic
  integer axis, dir, n1, n2, n3, k, ierr
  double precision u( n1, n2, n3 )
  integer i3, i2, i1, buff_len, buff_id
  buff_id = 2 + dir
  buff_len = 0
  if( axis .eq. 1 ) then
    if( dir .eq. -1 ) then
      do i3=2,n3-1
        do i2=2,n2-1
          do i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i3 )
          enddo
        enddo
      enddo
    else if( dir .eq. +1 ) then
      do i3=2,n3-1
        do i2=2,n2-1
          do i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1-1, i2, i3 )
          enddo
        enddo
      enddo
    endif
  else if( axis .eq. 2 ) then
    if( dir .eq. -1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i2-1 )
          enddo
        enddo
      enddo
    else if( dir .eq. +1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i2+1 )
          enddo
        enddo
      enddo
    endif
  else if( axis .eq. 3 ) then
    if( dir .eq. -1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i1-1 )
          enddo
        enddo
      enddo
    else if( dir .eq. +1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i1+1 )
          enddo
        enddo
      enddo
    endif
  endif
  return
end

subroutine take3( axis, dir, u, n1, n2, n3 )
  use caf_intrinsic
  implicit none
  include 'cafnpb.h'
  include 'globals.h'
  integer axis, dir, n1, n2, n3
  double precision u( n1, n2, n3 )
  integer buff_id, indx
  integer i3, i2, i1
  buff_id = 3 + dir
  indx = 0
  if( axis .eq. 1 ) then
    if( dir .eq. -1 ) then
      do i3=2,n3-1
        do i2=2,n2-1
          indx = indx + 1
          u(n1,i2,i3) = buff(indx, buff_id)
        enddo
      enddo
    else if( axis .eq. 2 ) then
      do i3=2,n3-1
        do i1=1,n1
          indx = indx + 1
          u(i1,i2,i3) = buff(indx, buff_id)
        enddo
      enddo
    else if( axis .eq. 3 ) then
      do i3=2,n3-1
        do i2=2,n2-1
          indx = indx + 1
          u(i1,i2,n3) = buff(indx, buff_id)
        enddo
      enddo
    endif
  endif
  return
end

subroutine commp( axis, u, n1, n2, n3, kk )
  use caf_intrinsic
  implicit none
  include 'cafnpb.h'
  include 'globals.h'
  integer axis, dir, n1, n2, n3
  double precision u( n1, n2, n3 )
  integer i3, i2, i1, buff_len, buff_id
  dir = -1
  buff_id = 3 + dir
  buff_len = 0
  if( axis .eq. 1 ) then
    if( dir .eq. -1 ) then
      do i3=2,n3-1
        do i2=2,n2-1
          do i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i3 )
          enddo
        enddo
      enddo
    else if( dir .eq. +1 ) then
      do i3=2,n3-1
        do i2=2,n2-1
          do i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1-1, i2, i3 )
          enddo
        enddo
      enddo
    endif
  else if( axis .eq. 2 ) then
    if( dir .eq. -1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i2-1 )
          enddo
        enddo
      enddo
    else if( dir .eq. +1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i2+1 )
          enddo
        enddo
      enddo
    endif
  else if( axis .eq. 3 ) then
    if( dir .eq. -1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i1-1 )
          enddo
        enddo
      enddo
    else if( dir .eq. +1 ) then
      do i3=2,n3-1
        do i1=1,n1
          do i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id) = u( i1, i2, i1+1 )
          enddo
        enddo
      enddo
    endif
  endif
  return
end
```




MPI Reality

```

1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  my $script = $0;
7  my $version = "1.0.0";
8  my $author = "John Doe";
9  my $email = "john.doe@example.com";
10
11 # Command line arguments
12 my $input = "input.txt";
13 my $output = "output.txt";
14 my $verbose = 0;
15
16 # Parse command line arguments
17 while (my $arg = shift) {
18     if ($arg eq "-i") {
19         $input = shift;
20     }
21     elsif ($arg eq "-o") {
22         $output = shift;
23     }
24     elsif ($arg eq "-v") {
25         $verbose = 1;
26     }
27     else {
28         die "Unknown option: $arg\n";
29     }
30 }
31
32 # Read input file
33 my $input_fh = open(my $fh, "<$input") or die "Cannot open $input for reading: $!";
34
35 # Process input
36 my $count = 0;
37 while (my $line = <$fh) {
38     $count++;
39     # Process each line (e.g., count words)
40     my $words = split(/\s+/, $line);
41     my $word_count = scalar(@words);
42
43     # Print word count for each line
44     print "$count: $word_count\n";
45 }
46
47 # Write output
48 my $output_fh = open(my $fh, ">$output") or die "Cannot open $output for writing: $!";
49 print $fh "$count\n";
50
51 # Close files
52 close($input_fh);
53 close($output_fh);
54
55 # Print summary
56 print "Total lines: $count\n";
57
58 # Exit
59 exit 0;

```

```

1  import { get, merge, mergeDeep } from 'lodash';
2
3  // ...
4
5  // ...
6
7  // ...
8
9  // ...
10
11  // ...
12
13  // ...
14
15  // ...
16
17  // ...
18
19  // ...
20
21  // ...
22
23  // ...
24
25  // ...
26
27  // ...
28
29  // ...
30
31  // ...
32
33  // ...
34
35  // ...
36
37  // ...
38
39  // ...
40
41  // ...
42
43  // ...
44
45  // ...
46
47  // ...
48
49  // ...
50
51  // ...
52
53  // ...
54
55  // ...
56
57  // ...
58
59  // ...
60
61  // ...
62
63  // ...
64
65  // ...
66
67  // ...
68
69  // ...
70
71  // ...
72
73  // ...
74
75  // ...
76
77  // ...
78
79  // ...
80
81  // ...
82
83  // ...
84
85  // ...
86
87  // ...
88
89  // ...
90
91  // ...
92
93  // ...
94
95  // ...
96
97  // ...
98
99  // ...
100
101  // ...
102
103  // ...
104
105  // ...
106
107  // ...
108
109  // ...
110
111  // ...
112
113  // ...
114
115  // ...
116
117  // ...
118
119  // ...
120
121  // ...
122
123  // ...
124
125  // ...
126
127  // ...
128
129  // ...
130
131  // ...
132
133  // ...
134
135  // ...
136
137  // ...
138
139  // ...
140
141  // ...
142
143  // ...
144
145  // ...
146
147  // ...
148
149  // ...
150
151  // ...
152
153  // ...
154
155  // ...
156
157  // ...
158
159  // ...
160
161  // ...
162
163  // ...
164
165  // ...
166
167  // ...
168
169  // ...
170
171  // ...
172
173  // ...
174
175  // ...
176
177  // ...
178
179  // ...
180
181  // ...
182
183  // ...
184
185  // ...
186
187  // ...
188
189  // ...
190
191  // ...
192
193  // ...
194
195  // ...
196
197  // ...
198
199  // ...
200
201  // ...
202
203  // ...
204
205  // ...
206
207  // ...
208
209  // ...
210
211  // ...
212
213  // ...
214
215  // ...
216
217  // ...
218
219  // ...
220
221  // ...
222
223  // ...
224
225  // ...
226
227  // ...
228
229  // ...
230
231  // ...
232
233  // ...
234
235  // ...
236
237  // ...
238
239  // ...
240
241  // ...
242
243  // ...
244
245  // ...
246
247  // ...
248
249  // ...
250
251  // ...
252
253  // ...
254
255  // ...
256
257  // ...
258
259  // ...
260
261  // ...
262
263  // ...
264
265  // ...
266
267  // ...
268
269  // ...
270
271  // ...
272
273  // ...
274
275  // ...
276
277  // ...
278
279  // ...
280
281  // ...
282
283  // ...
284
285  // ...
286
287  // ...
288
289  // ...
290
291  // ...
292
293  // ...
294
295  // ...
296
297  // ...
298
299  // ...
300
301  // ...
302
303  // ...
304
305  // ...
306
307  // ...
308
309  // ...
310
311  // ...
312
313  // ...
314
315  // ...
316
317  // ...
318
319  // ...
320
321  // ...
322
323  // ...
324
325  // ...
326
327  // ...
328
329  // ...
330
331  // ...
332
333  // ...
334
335  // ...
336
337  // ...
338
339  // ...
340
341  // ...
342
343  // ...
344
345  // ...
346
347  // ...
348
349  // ...
350
351  // ...
352
353  // ...
354
355  // ...
356
357  // ...
358
359  // ...
360
361  // ...
362
363  // ...
364
365  // ...
366
367  // ...
368
369  // ...
370
371  // ...
372
373  // ...
374
375  // ...
376
377  // ...
378
379  // ...
380
381  // ...
382
383  // ...
384
385  // ...
386
387  // ...
388
389  // ...
390
391  // ...
392
393  // ...
394
395  // ...
396
397  // ...
398
399  // ...
400
401  // ...
402
403  // ...
404
405  // ...
406
407  // ...
408
409  // ...
410
411  // ...
412
413  // ...
414
415  // ...
416
417  // ...
418
419  // ...
420
421  // ...
422
423  // ...
424
425  // ...
426
427  // ...
428
429  // ...
430
431  // ...
432
433  // ...
434
435  // ...
436
437  // ...
438
439  // ...
440
441  // ...
442
443  // ...
444
445  // ...
446
447  // ...
448
449  // ...
450
451  // ...
452
453  // ...
454
455  // ...
456
457  // ...
458
459  // ...
460
461  // ...
462
463  // ...
464
465  // ...
466
467  // ...
468
469  // ...
470
471  // ...
472
473  // ...
474
475  // ...
476
477  // ...
478
479  // ...
480
481  // ...
482
483  // ...
484
485  // ...
486
487  // ...
488
489  // ...
490
491  // ...
492
493  // ...
494
495  // ...
496
497  // ...
498
499  // ...
500
501  // ...
502
503  // ...
504
505  // ...
506
507  // ...
508
509  // ...
510
511  // ...
512
513  // ...
514
515  // ...
516
517  // ...
518
519  // ...
520
521  // ...
522
523  // ...
524
525  // ...
526
527  // ...
528
529  // ...
530
531  // ...
532
533  // ...
534
535  // ...
536
537  // ...
538
539  // ...
540
541  // ...
542
543  // ...
544
545  // ...
546
547  // ...
548
549  // ...
550
551  // ...
552
553  // ...
554
555  // ...
556
557  // ...
558
559  // ...
560
561  // ...
562
563  // ...
564
565  // ...
566
567  // ...
568
569  // ...
570
571  // ...
572
573  // ...
574
575  // ...
576
577  // ...
578
579  // ...
580
581  // ...
582
583  // ...
584
585  // ...
586
587  // ...
588
589  // ...
590
591  // ...
592
593  // ...
594
595  // ...
596
597  // ...
598
599  // ...
600
601  // ...
602
603  // ...
604
605  // ...
606
607  // ...
608
609  // ...
610
611  // ...
612
613  // ...
614
615  // ...
616
617  // ...
618
619  // ...
620
621  // ...
622
623  // ...
624
625  // ...
626
627  // ...
628
629  // ...
630
631  // ...
632
633  // ...
634
635  // ...
636
637  // ...
638
639  // ...
640
641  // ...
642
643  // ...
644
645  // ...
646
647  // ...
648
649  // ...
650
651  // ...
652
653  // ...
654
655  // ...
656
657  // ...
658
659  // ...
660
661  // ...
662
663  // ...
664
665  // ...
666
667  // ...
668
669  // ...
670
671  // ...
672
673  // ...
674
675  // ...
676
677  // ...
678
679  // ...
680
681  // ...
682
683  // ...
684
685  // ...
686
687  // ...
688
689  // ...
690
691  // ...
692
693  // ...
694
695  // ...
696
697  // ...
698
699  // ...
700
701  // ...
702
703  // ...
704
705  // ...
706
707  // ...
708
709  // ...
710
711  // ...
712
713  // ...
714
715  // ...
716
717  // ...
718
719  // ...
720
721  // ...
722
723  // ...
724
725  // ...
726
727  // ...
728
729  // ...
730
731  // ...
732
733  // ...
734
735  // ...
736
737  // ...
738
739  // ...
740
741  // ...
742
743  // ...
744
745  // ...
746
747  // ...
748
749  // ...
750
751  // ...
752
753  // ...
754
755  // ...
756
757  // ...
758
759  // ...
760
761  // ...
762
```

[illegible]

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include <unistd.h>
7  #include <sys/types.h>
8  #include <sys/stat.h>
9  #include <fcntl.h>
10 #include <sys/time.h>
11 #include <sys/mman.h>
12 #include <sys/wait.h>
13 #include <sys/resource.h>
14 #include <sys/param.h>
15 #include <sys/uio.h>
16 #include <sys/errno.h>
17 #include <sys/sysmacros.h>
18 #include <sys/unistd.h>
19 #include <sys/mount.h>
20 #include <sys/procfs.h>
21 #include <sys/proc.h>
22 #include <sys/procfs.h>
23 #include <sys/procfs.h>
24 #include <sys/procfs.h>
25 #include <sys/procfs.h>
26 #include <sys/procfs.h>
27 #include <sys/procfs.h>
28 #include <sys/procfs.h>
29 #include <sys/procfs.h>
30 #include <sys/procfs.h>
31 #include <sys/procfs.h>
32 #include <sys/procfs.h>
33 #include <sys/procfs.h>
34 #include <sys/procfs.h>
35 #include <sys/procfs.h>
36 #include <sys/procfs.h>
37 #include <sys/procfs.h>
38 #include <sys/procfs.h>
39 #include <sys/procfs.h>
40 #include <sys/procfs.h>
41 #include <sys/procfs.h>
42 #include <sys/procfs.h>
43 #include <sys/procfs.h>
44 #include <sys/procfs.h>
45 #include <sys/procfs.h>
46 #include <sys/procfs.h>
47 #include <sys/procfs.h>
48 #include <sys/procfs.h>
49 #include <sys/procfs.h>
50 #include <sys/procfs.h>
51 #include <sys/procfs.h>
52 #include <sys/procfs.h>
53 #include <sys/procfs.h>
54 #include <sys/procfs.h>
55 #include <sys/procfs.h>
56 #include <sys/procfs.h>
57 #include <sys/procfs.h>
58 #include <sys/procfs.h>
59 #include <sys/procfs.h>
60 #include <sys/procfs.h>
61 #include <sys/procfs.h>
62 #include <sys/procfs.h>
63 #include <sys/procfs.h>
64 #include <sys/procfs.h>
65 #include <sys/procfs.h>
66 #include <sys/procfs.h>
67 #include <sys/procfs.h>
68 #include <sys/procfs.h>
69 #include <sys/procfs.h>
70 #include <sys/procfs.h>
71 #include <sys/procfs.h>
72 #include <sys/procfs.h>
73 #include <sys/procfs.h>
74 #include <sys/procfs.h>
75 #include <sys/procfs.h>
76 #include <sys/procfs.h>
77 #include <sys/procfs.h>
78 #include <sys/procfs.h>
79 #include <sys/procfs.h>
80 #include <sys/procfs.h>
81 #include <sys/procfs.h>
82 #include <sys/procfs.h>
83 #include <sys/procfs.h>
84 #include <sys/procfs.h>
85 #include <sys/procfs.h>
86 #include <sys/procfs.h>
87 #include <sys/procfs.h>
88 #include <sys/procfs.h>
89 #include <sys/procfs.h>
90 #include <sys/procfs.h>
91 #include <sys/procfs.h>
92 #include <sys/procfs.h>
93 #include <sys/procfs.h>
94 #include <sys/procfs.h>
95 #include <sys/procfs.h>
96 #include <sys/procfs.h>
97 #include <sys/procfs.h>
98 #include <sys/procfs.h>
99 #include <sys/procfs.h>
100 #include <sys/procfs.h>

```

[illegible][illegible][illegible][illegible][illegible]

```

1  // Computing the number of elements in the subarray [l, r]
2  // that are greater than or equal to x.
3  int countGreaterOrEqual(int x, int l, int r) {
4      int count = 0;
5      for (int i = l; i <= r; i++) {
6          if (arr[i] >= x) count++;
7      }
8      return count;
9  }
10
11 // Finding the median of the array.
12 int findMedian(int arr[], int n) {
13     int l = 0, r = n - 1;
14     while (l < r) {
15         int mid = (l + r) / 2;
16         int count = countGreaterOrEqual(arr[mid], l, r);
17         if (count < (n + 1) / 2) l = mid + 1;
18         else r = mid;
19     }
20     return arr[l];
21 }
22
23 // Driver code
24 int main() {
25     int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
26     int n = sizeof(arr) / sizeof(arr[0]);
27     int median = findMedian(arr, n);
28     cout << "Median of the array is: " << median << endl;
29     return 0;
30 }

```

[illegible]

- New functional.
- Bonded systems.
- 552 lines C code.

WJDC-DFT (Werthim, Jain, Dominik, and Chapman) theory for bonded systems. (S. Jain, A. Dominik, and W.G. Chapman. *Modified interfacial statistical associating fluid theory: A perturbation density functional theory for inhomogeneous complex fluids*. *J. Chem. Phys.*, 127:244904, 2007.) Models stoichiometry constraints inherent to bonded systems.

How much MPI-specific code?

dft_fill_wjdc.c
MPI-specific
code

source_pp_g.f[illegible][illegible][illegible][illegible]

```

A_MU[P_K, j, LOCAL_ID] = ZERO
A_MU[P_K, j, LOCAL_ID] = ZERO
A_MU[P_K, j, LOCAL_ID] = ZERO
A_MU[P_K, j, LOCAL_ID] = ZERO
A_MU[P_K, j, LOCAL_ID] = ZERO
A_MU[P_K, j, LOCAL_ID] = ZERO
A_MU[P_K, j, LOCAL_ID] = ONE
A_MU[P_K, j, LOCAL_ID] = ZERO
ENDIF
ENDIF

CUMM = S(ART, base)
IF CAL(1) == CAL, S(ART, THEN
SUM_R_0 = SUM_R_0 + sum
SUM_R_1 = SUM_R_1 + sum
ENDIF
ENDIF
CUMM = S(ART, end)
call collect_val_array

RETURN
END SUBROUTINE SOURCE_PP_0

// Comments on the modifications for DMP versus implementation
// 001 Include header file and common declarations for parallelization
// 002 Change of loop limit, i, loopend2 = i, loopend3, i, loopend4

```

MFIX

Source term for
pressure
correction

- MPI-callable, OpenMP-enabled.
- 340 Fortran lines.
- No MPI-specific code.
- Ubiquitous OpenMP markup (red regions).

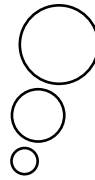
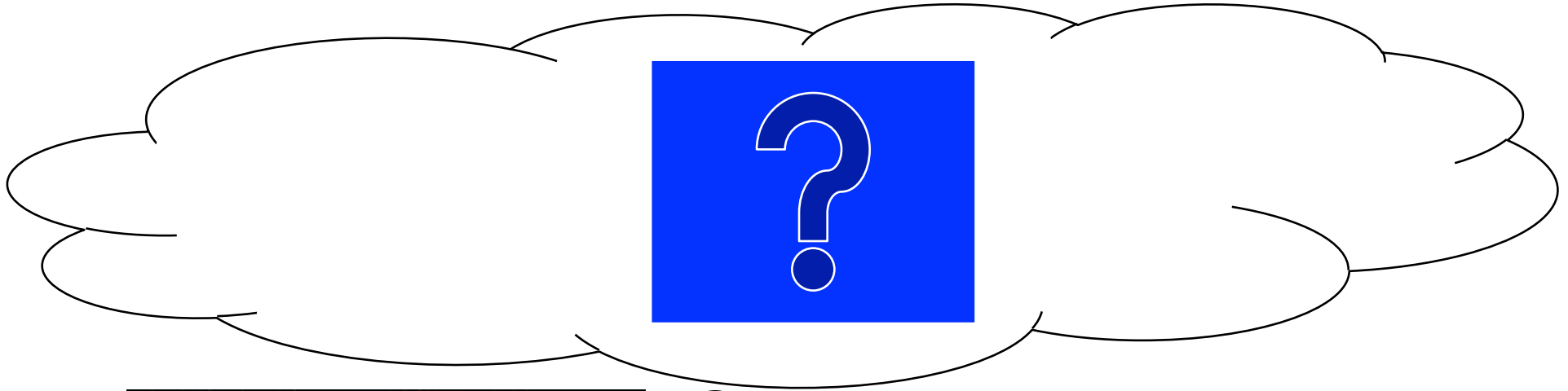


Reasons for MPI Success?

- Portability? Yes.
- Standardized? Yes.
- Momentum? Yes.
- Separation of many Parallel & Algorithms concerns? Big Yes.
- Once framework in place:
 - Sophisticated physics added as serial code.
 - Ratio of science experts vs. parallel experts: 10:1.
- Key goal for new parallel apps: Preserve this ratio



Computational Domain Expert Writing Future Parallel Code

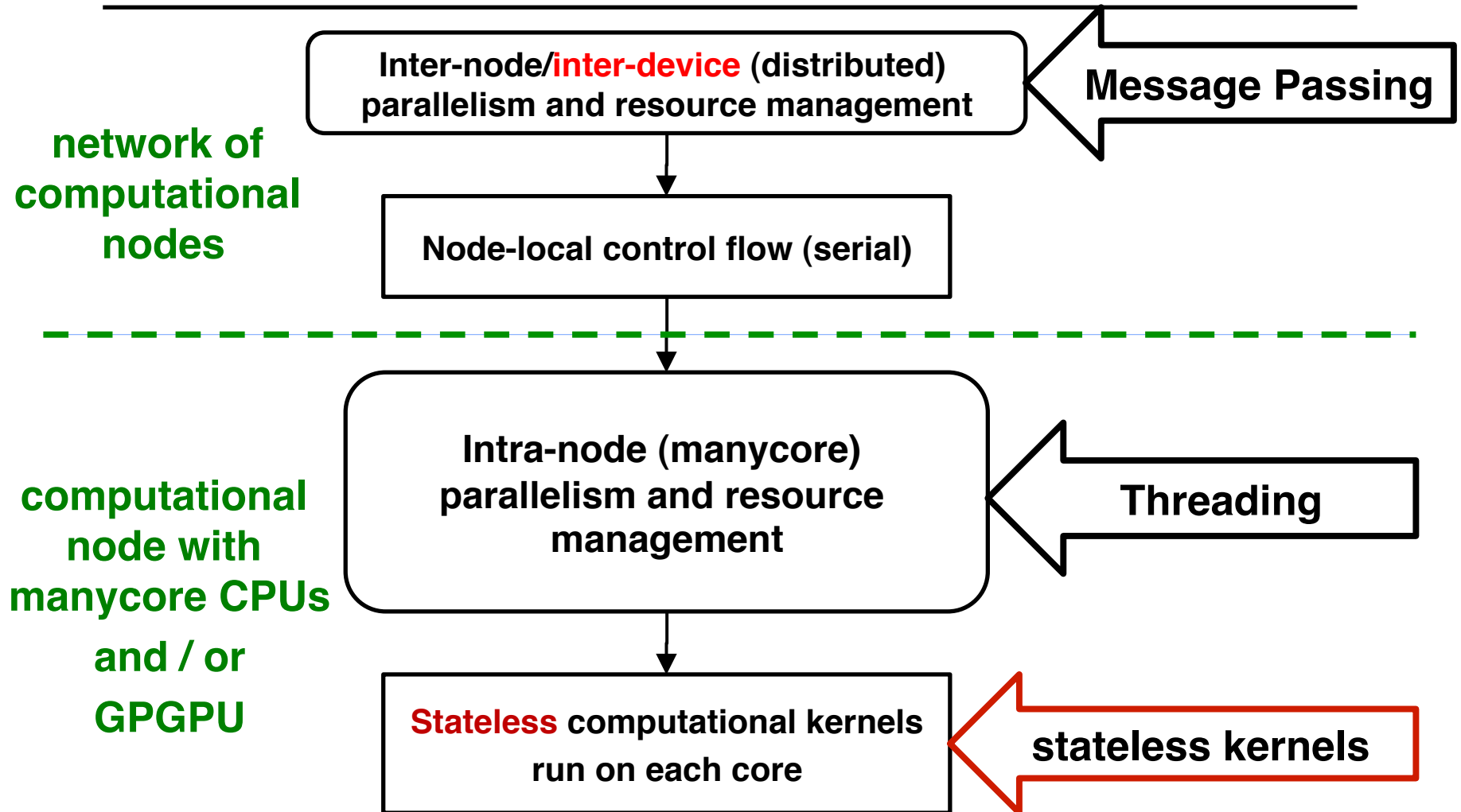




Evolving Parallel Programming Model



Parallel Programming Model: Multi-level/Multi-device





Domain Scientist's Parallel Palette

- MPI-only (SPMD) apps:
 - Single parallel construct.
 - Simultaneous execution.
 - Parallelism of even the messiest serial code.
- MapReduce:
 - Plug-n-Play data processing framework - 80% Google cycles.
- Pregel: Graph framework (other 20%)
- Next-generation PDE and related applications:
 - Internode:
 - MPI, yes, or something like it.
 - Composed with intranode.
 - Intranode:
 - Much richer palette.
 - More care required from programmer.
- What are the constructs in our new palette?



Obvious Constructs/Concerns

- Parallel for:
 - No loop-carried dependence.
 - Rich loops.
 - Use of shared memory for temporal reuse, efficient device data transfers.
- Parallel reduce:
 - Couple with other computations.
 - Concern for reproducibility.

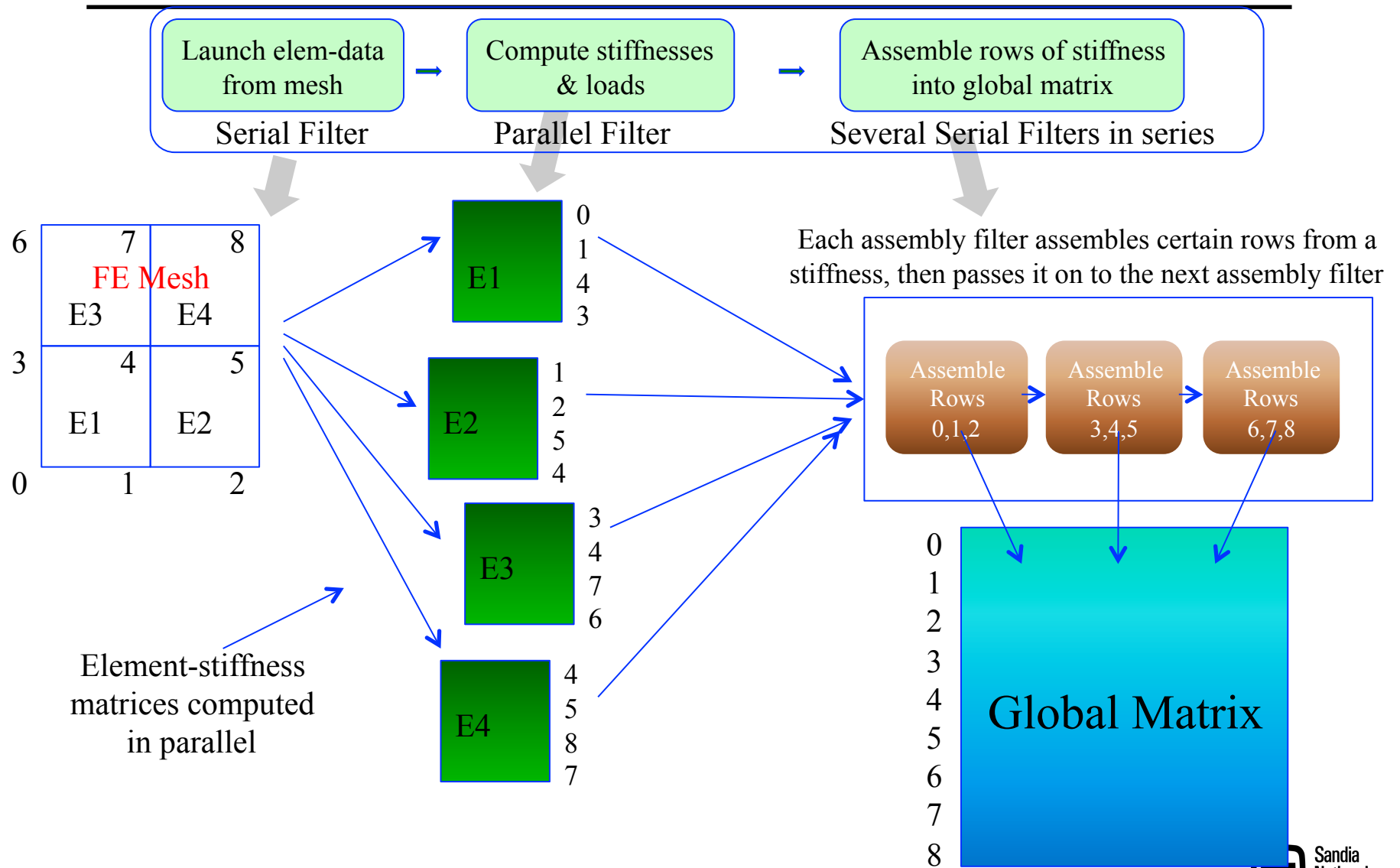


Other construct: Pipeline

- Sequence of filters.
- Each filter is:
 - Sequential (grab element ID, enter global assembly) or
 - Parallel (fill element stiffness matrix).
- Filters executed in sequence.
- Programmer's concern:
 - Determine (conceptually): Can filter execute in parallel?
 - Write filter (serial code).
 - Register it with the pipeline.
- Extensible:
 - New physics feature.
 - New filter added to pipeline.

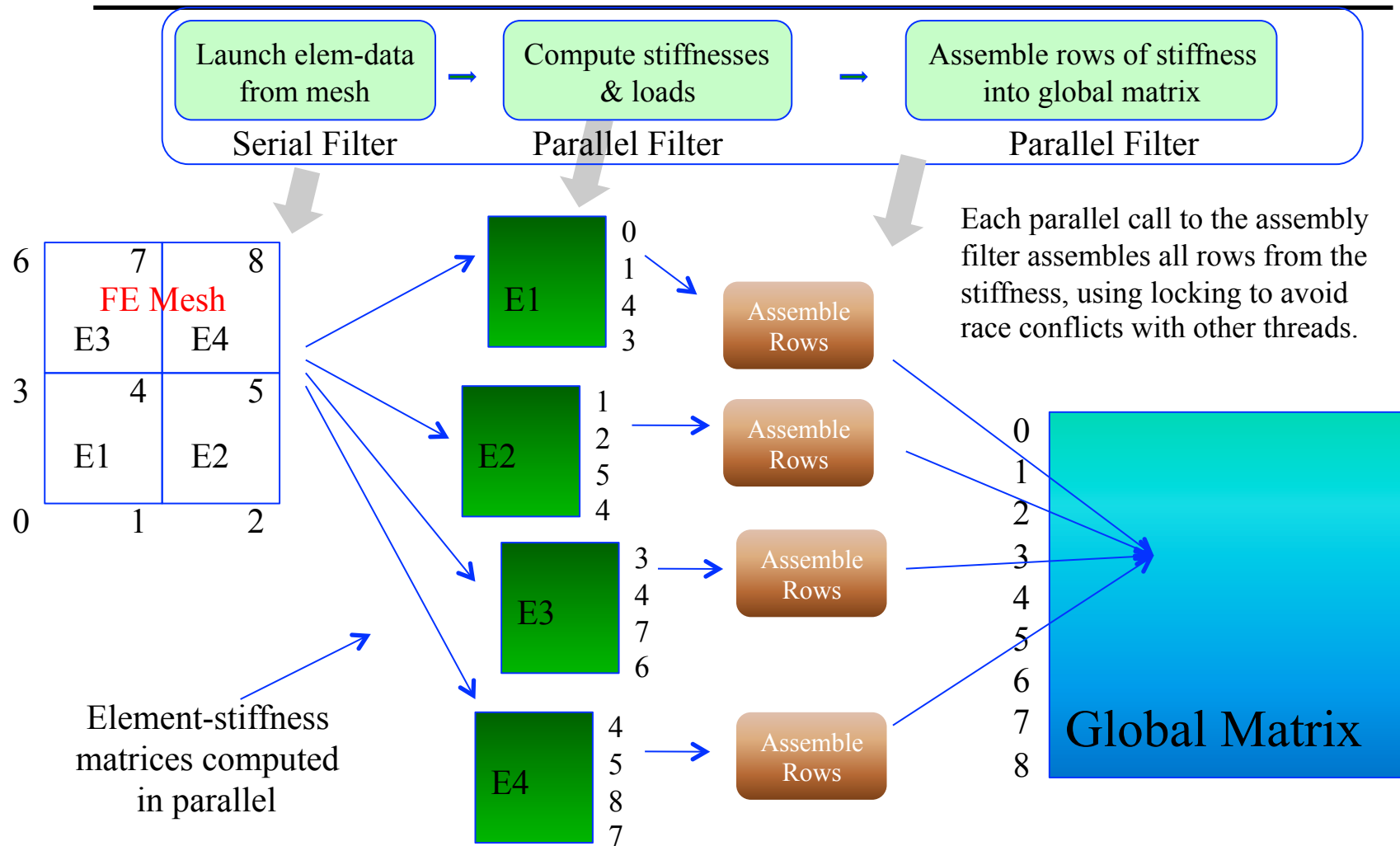


TBB Pipeline for FE assembly





Alternative TBB Pipeline for FE assembly





Base-line FE Assembly Timings

Problem size: 80x80x80 == 512000 elements, 531441 matrix-rows

The finite-element assembly performs 4096000 matrix-row sum-into operations (8 per element) and 4096000 vector-entry sum-into operations.

MPI-only, no threads. Linux dual quad-core workstation.

Num-procs	Assembly-time Intel 11.1	Assembly-time GCC 4.4.4
1	1.80s	1.95s
4	0.45s	0.50s
8	0.24s	0.28s

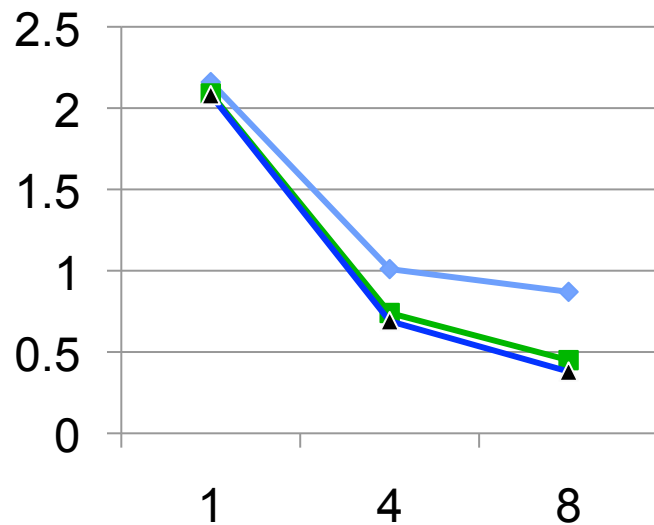


FE Assembly Timings

Problem size: 80x80x80 == 512000 elements, 531441 matrix-rows

The finite-element assembly performs 4096000 matrix-row sum-into operations (8 per element) and 4096000 vector-entry sum-into operations.

No MPI, only threads. Linux dual quad-core workstation.



Num-threads	Elem-group-size	Matrix-conflicts	Vector-conflicts	Assembly-time
1	1	0	0	2.16s
1	4	0	0	2.09s
1	8	0	0	2.08s
4	1	95917	959	1.01s
4	4	7938	25	0.74s
4	8	3180	4	0.69s
8	1	64536	1306	0.87s
8	4	5892	49	0.45s
8	8	1618	1	0.38s

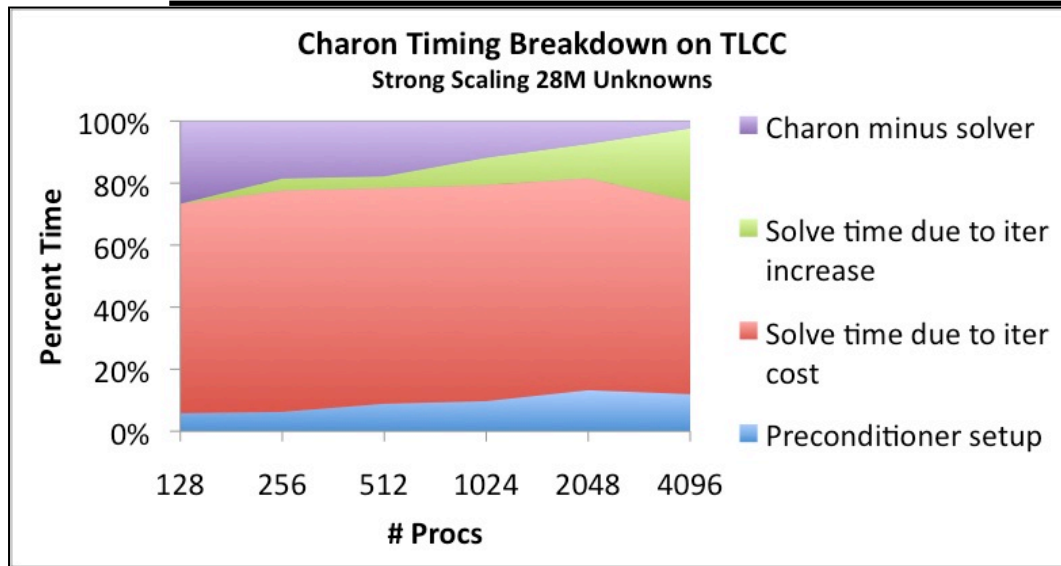


Other construct: Thread team

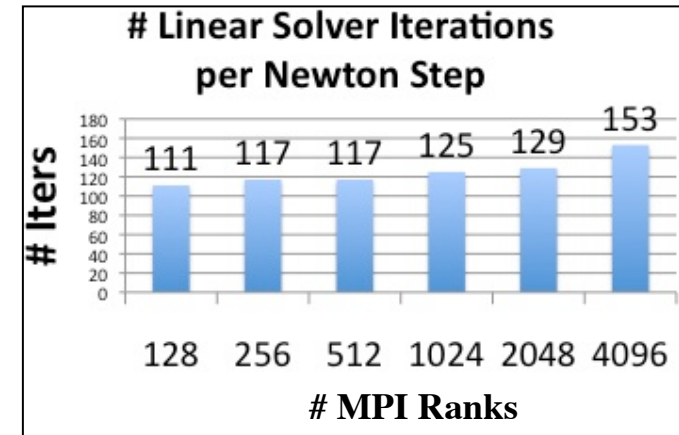
- Multiple threads.
- Fast barrier.
- Shared, fast access memory pool.
- Example: Nvidia SM
- X86 more vague, emerging more clearly in future.



Preconditioners for Scalable Multicore Systems



Strong scaling of Charon on TLCC (P. Lin, J. Shadid 2009)



- Observe: Iteration count increases with number of subdomains.
- With scalable threaded smoothers (LU, ILU, Gauss-Seidel):
 - Solve with fewer, larger subdomains.
 - Better kernel scaling (threads vs. MPI processes).
 - Better convergence, More robust.
- Exascale Potential: Tiled, pipelined implementation.
- **Three efforts:**
 - Level-scheduled triangular sweeps (ILU solve, Gauss-Seidel).
 - **Decomposition by partitioning**
 - Multithreaded direct **factorization**

MPI Tasks	Threads	Iterations
4096	1	153
2048	2	129
1024	4	125
512	8	117
256	16	117
128	32	111

Factors Impacting Performance of Multithreaded Sparse Triangular Solve, Michael M. Wolf and Michael A. Heroux and Erik G. Boman, VECPAR 2010.



Thread Team Advantages

- Qualitatively better algorithm:
 - Threaded triangular solve scales.
 - Fewer MPI ranks means fewer iterations, better robustness.
- Exploits:
 - Shared data.
 - Fast barrier.
 - Data-driven parallelism.



Finite Elements/Volumes/Differences and parallel node constructs

- Parallel for, reduce, pipeline:
 - Sufficient for vast majority of node level computation.
 - Supports:
 - Complex modeling expression.
 - Vanilla parallelism.
 - Must be “stencil-aware” for temporal locality.
- Thread team:
 - Complicated.
 - Requires true parallel algorithm knowledge.
 - Useful in solvers.



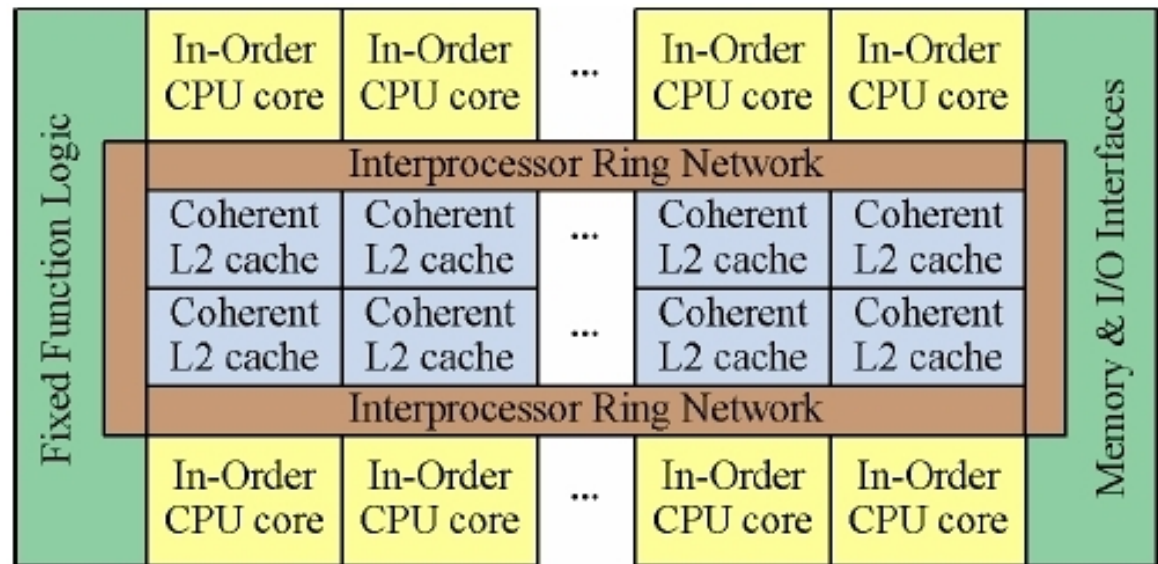
Portable Multi/Manycore Programming Trilinos/Kokkos Node API



Another Manycore architecture: Intel MIC

Knights Ferry:

- 32 x86 cores
 - 4-way hyperthreading
 - 128 threads total
- 512-bit vector unit
 - 16 floats, 8 doubles
- 1.20GHz
- PCI-E 2.0
- 2GB GDDR5 global mem
 - 8MB shared L2 cache
 - 64KB L1 Data, 64KB L1 Inst



Programming Env:

- OpenMP,
- TBB,
- Pthreads

Tpetra and Kokkos

- **Tpetra** is an implementation of the Petra Object Model.
 - Design is similar to Epetra, with appropriate deviation.
 - Fundamental differences:
 - heavily exploits templates
 - utilizes hybrid (distributed + **shared**) parallelism via Kokkos Node API
- **Kokkos** is an API for shared-memory parallel nodes
 - Provides `parallel_for` and `parallel_reduce` skeletons.
 - Support shared memory APIs:
 - ThreadPool Interface (TPI; Carter Edwards's pthreads Trilinos package)
 - Intel Threading Building Blocks (TBB)
 - NVIDIA CUDA-capable GPUs (via Thrust)
 - *OpenMP (implemented by Radu Popescu/EPFL, awaiting my git push)*



Generic Shared Memory Node

- Abstract inter-node comm provides DMP support.
- Need some way to **portably** handle SMP support.
- Goal: allow code, once written, to be run on **any parallel node**, regardless of architecture.
- **Difficulty #1**: Many different **memory architectures**
 - Node may have multiple, disjoint memory spaces.
 - Optimal performance may require special memory placement.
- **Difficulty #2**: **Kernels** must be tailored to architecture
 - Implementation of optimal kernel will vary between archs
 - No universal binary → need for separate compilation paths
- Practical goal: Cover 80% kernels with generic code.



Kokkos Node API

- Kokkos provides two main components:
 - Kokkos memory model addresses Difficulty #1
 - Allocation, deallocation and efficient access of memory
 - compute buffer: special memory used for parallel computation
 - New: Local Store Pointer and Buffer with size.
 - Kokkos compute model addresses Difficulty #2
 - Description of kernels for parallel execution on a node
 - Provides stubs for common parallel work constructs
 - Currently, parallel for loop and parallel reduce
- Code is developed around a polymorphic Node object.
- Supporting a new platform requires only the implementation of a new node type.



Kokkos Memory Model

- A generic node model must at least:
 - support the scenario involving **distinct device memory**
 - allow **efficient** memory access under traditional scenarios
- Nodes provide the following memory routines:

```
ArrayRCP<T> Node::allocBuffer<T>(size_t sz);  
void        Node::copyToBuffer<T>( T * src,  
                                   ArrayRCP<T> dest);  
void        Node::copyFromBuffer<T>(ArrayRCP<T> src,  
                                   T * dest);  
ArrayRCP<T> Node::viewBuffer<T> (ArrayRCP<T> buff);  
void        Node::readyBuffer<T>(ArrayRCP<T> buff);
```



Kokkos Compute Model

- How to make shared-memory programming generic:
 - **Parallel reduction** is the intersection of `dot()` and `norm1()`
 - **Parallel for loop** is the intersection of `axpy()` and mat-vec
 - We need a way of **fusing** kernels with these basic **constructs**.
- Template meta-programming is **the answer**.
 - This is the same approach that Intel TBB and Thrust take.
 - Has the effect of requiring that Tpetra objects be templated on Node type.
- Node provides generic parallel constructs, user fills in the rest:

```
template <class WDP>
void Node::parallel_for(
    int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- loop body via `WDP::execute(i)`

```
template <class WDP>
WDP::ReductionType Node::parallel_reduce(
    int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- reduction type `WDP::ReductionType`
- element generation via `WDP::generate(i)`
- reduction via `WDP::reduce(x, y)`



Example Kernels: `axpy()` and `dot()`

```
template <class WDP>
void
Node::parallel_for(int beg, int end,
                   WDP workdata    );
```

```
template <class WDP>
WDP::ReductionType
Node::parallel_reduce(int beg, int end,
                     WDP workdata    );
```

```
template <class T>
struct AxyOp {
    const T * x;
    T * y;
    T alpha, beta;
    void execute(int i)
    { y[i] = alpha*x[i] + beta*y[i]; }
};
```

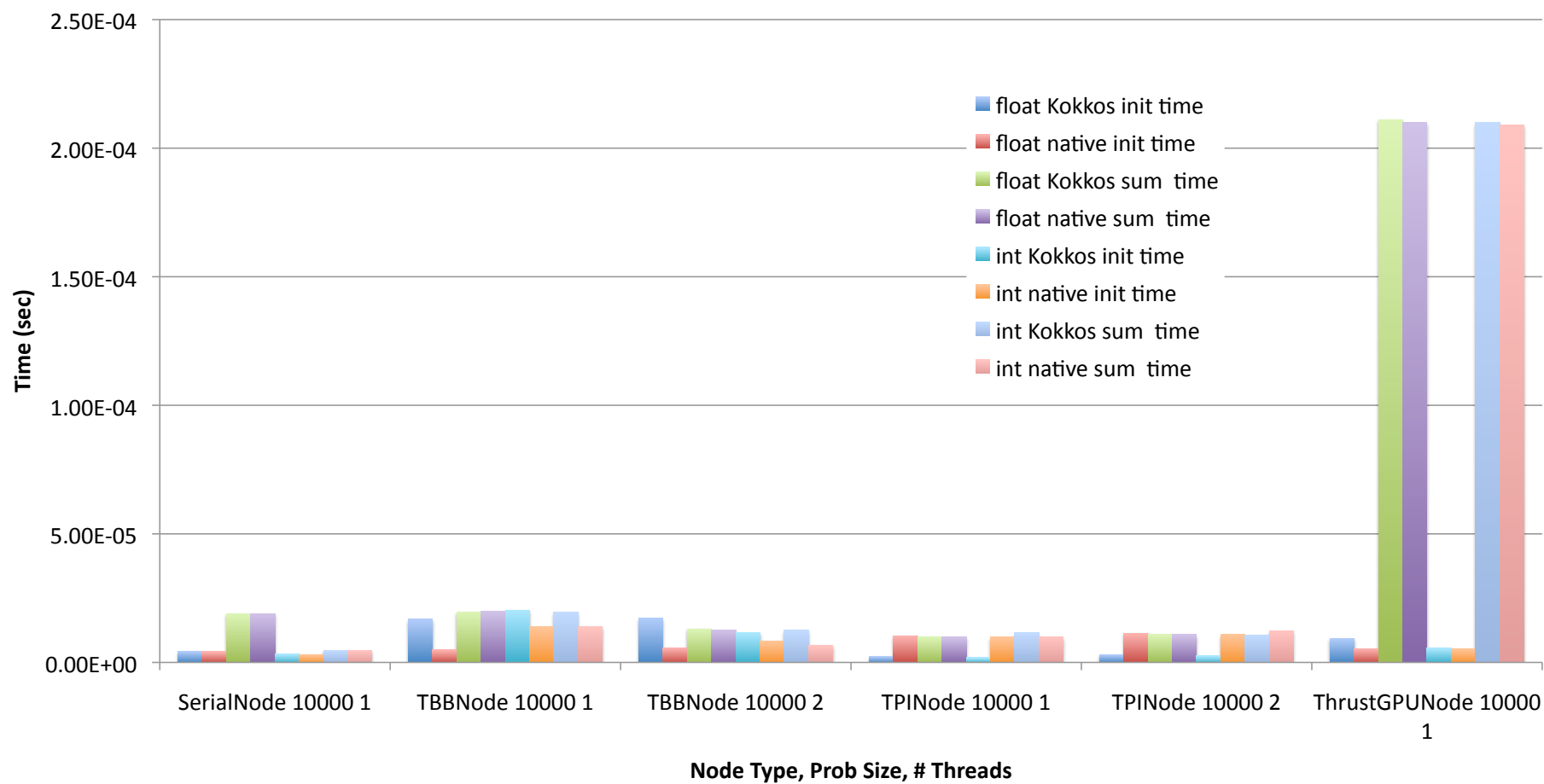
```
template <class T>
struct DotOp {
    typedef T ReductionType;
    const T * x, * y;
    T identity()      { return (T)0;      }
    T generate(int i) { return x[i]*y[i]; }
    T reduce(T x, T y) { return x + y;    }
};
```

```
AxyOp<double> op;
op.x = ...; op.alpha = ...;
op.y = ...; op.beta  = ...;
node.parallel_for< AxyOp<double> >
    (0, length, op);
```

```
DotOp<float> op;
op.x = ...; op.y = ...;
float dot;
dot = node.parallel_reduce< DotOp<float> >
    (0, length, op);
```

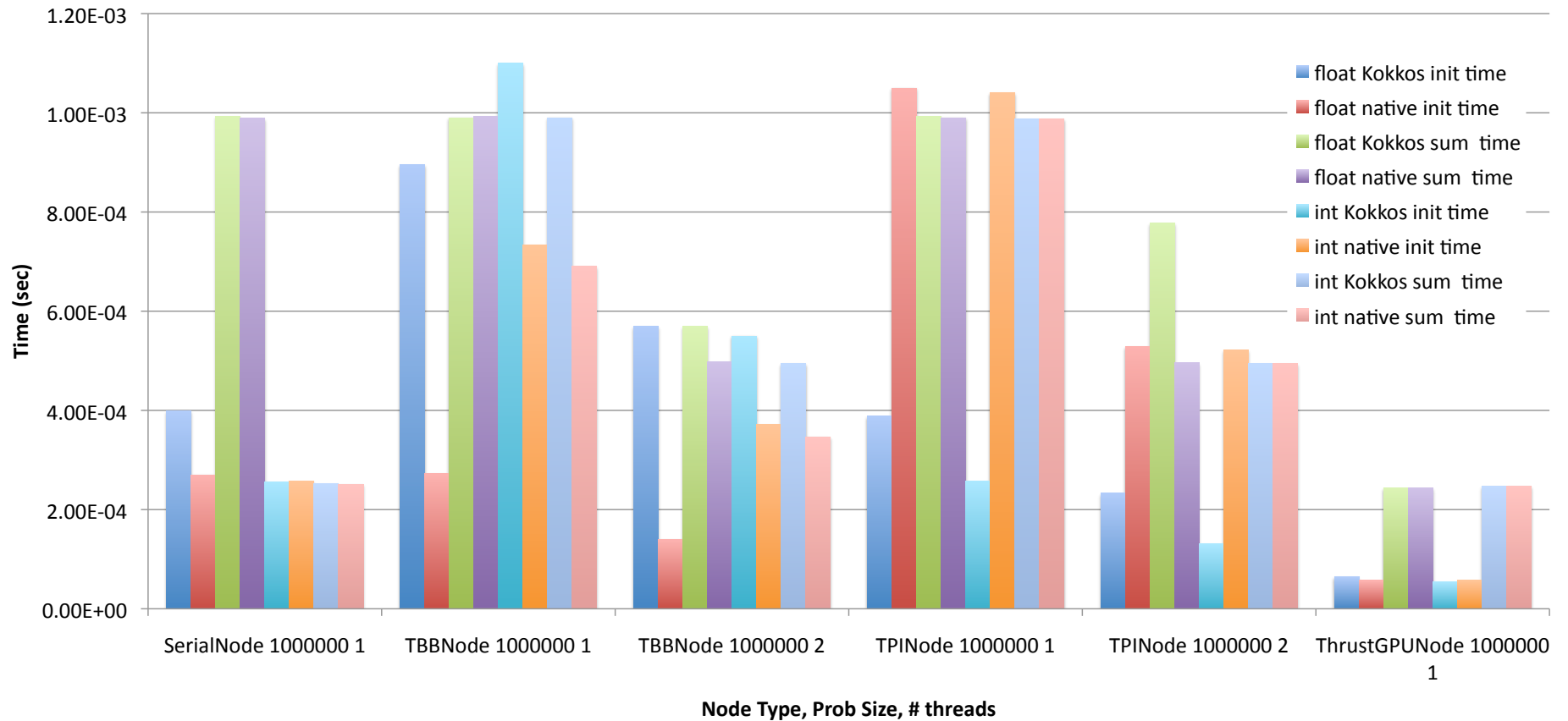


Kokkos Node API vs Native Implementation Axy, len=10K, float, int data





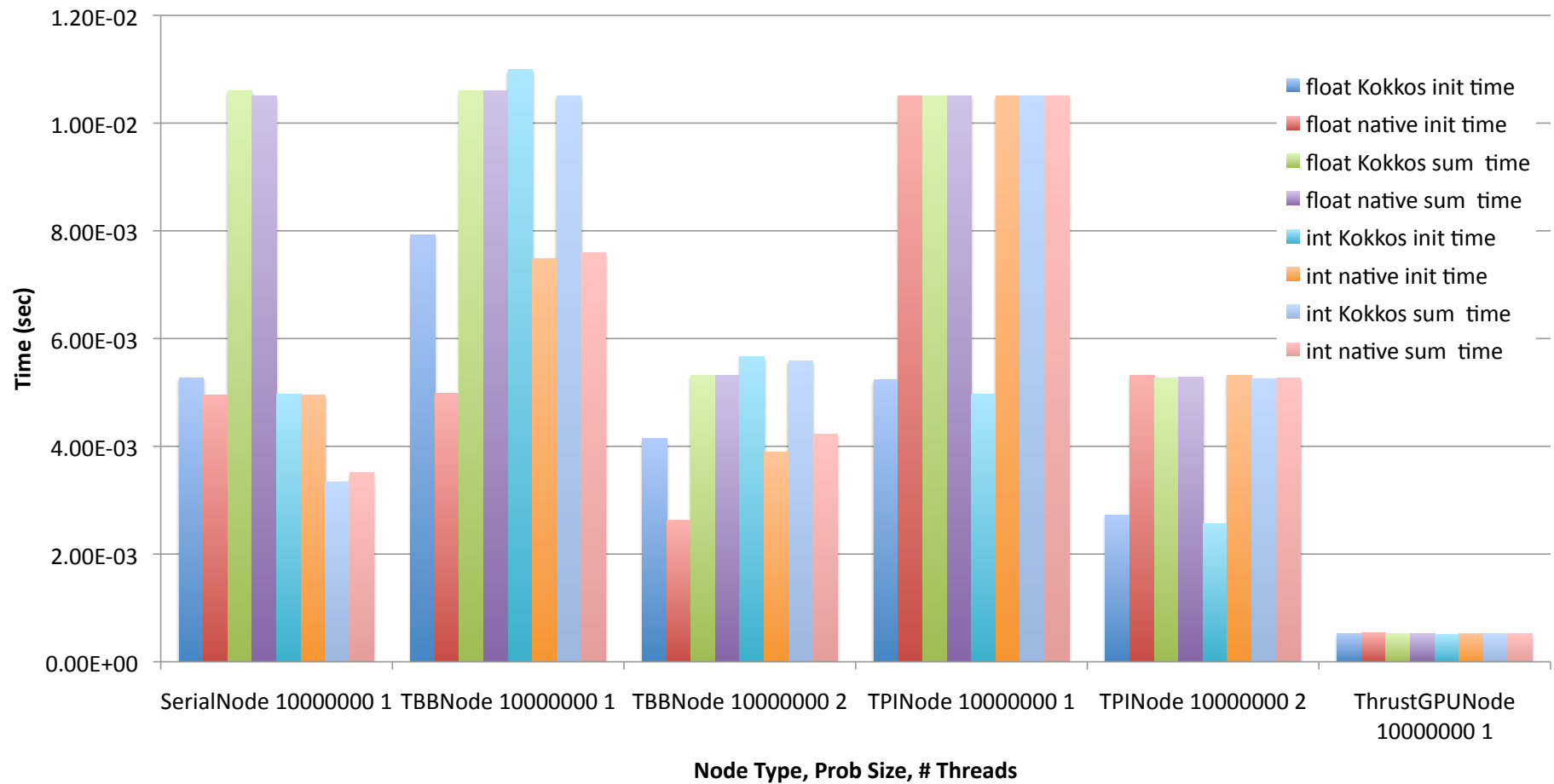
Kokkos Node API vs Native Implementation Axy, len=1M





Kokkos Node API vs Native Implementation

Axpy, len=10M, float, int data



What's the Big Deal about Vector-Vector Operations?

Examples from OOQP (Gertz, Wright)

$$y_i \leftarrow y_i + \alpha x_i z_i, i = 1 \dots n$$

$$y_i \leftarrow y_i / x_i, i = 1 \dots n$$

$$y_i \leftarrow \begin{cases} y^{\min} - y_i & \text{if } y_i < y^{\min} \\ y^{\max} - y_i & \text{if } y_i > y^{\max} \\ 0 & \text{if } y^{\min} \leq y_i \leq y^{\max} \end{cases}, i = 1 \dots n$$

$$\alpha \leftarrow \{ \max \alpha : x + \alpha d \geq \beta \}$$

Example from TRICE (Dennis, Heinkenschloss, Vicente)

$$d_i \leftarrow \begin{cases} (b - u)_i^{1/2} & \text{if } w_i < 0 \text{ and } b_i < +\infty \\ 1 & \text{if } w_i < 0 \text{ and } b_i = +\infty \\ (u - a)_i^{1/2} & \text{if } w_i \geq 0 \text{ and } a_i > -\infty \\ 1 & \text{if } w_i \geq 0 \text{ and } a_i = -\infty \end{cases}, i = 1 \dots n$$

Many different and unusual vector operations are needed by interior point methods for optimization!

Example from IPOPT (Wächter)

$$x_i \leftarrow \begin{cases} \left(x_i^L + \frac{(x_i^U - x_i^L)}{2} \right) & \text{if } \hat{x}_i^L > \hat{x}_i^U \\ \hat{x}_i^L & \text{if } x_i < \hat{x}_i^L \\ \hat{x}_i^U & \text{if } x_i > \hat{x}_i^U \end{cases}, i = 1 \dots n$$

$$\text{where: } \begin{aligned} \hat{x}_i^L &= \min \left(x_i^L + \eta (x_i^U - x_i^L) x_i^L + \delta \right) \\ \hat{x}_i^U &= \max \left(x_i^L - \eta (x_i^U - x_i^L) x_i^U - \delta \right) \end{aligned}$$

Currently in MOOCHO :
> 40 vector operations!

Tpetra RTI Components

- Set of stand-alone non-member methods:

- `unary_transform<UOP>(Vector &v, UOP op)`
- `binary_transform<BOP>(Vector &v1, const Vector &v2, BOP op)`
- `reduce<G>(const Vector &v1, const Vector &v2, G op_glob)`
- `binary_pre_transform_reduce<G>(Vector &v1,
const Vector &v2,
G op_glob)`

- These are non-member methods of `Tpetra::RTI` which are loosely coupled with `Tpetra::MultiVector` and `Tpetra::Vector`.

- `Tpetra::RTI` also provides Operator-wrappers:

- `class KernelOp<..., Kernel > : Tpetra::Operator<...>`
- `class BinaryOp<..., BinaryOp> : Tpetra::Operator<...>`

Tpetra RTI Example

[illegible]

Multiprecision possibilities

- **Tpetra** is a templated version of the Petra distributed linear algebra model in Trilinos.

- ◆ Objects are templated on the underlying data types:

```
MultiVector<scalar=double, local_ordinal=int,  
            global_ordinal=local_ordinal> ...  
CrsMatrix<scalar=double, local_ordinal=int,  
          global_ordinal=local_ordinal> ...
```

- ◆ Examples:

```
MultiVector<double, int, long int> V;  
CrsMatrix<float> A;
```

Speedup of float over double
in Belos linear solver.

float	double	speedup
18 s	26 s	1.42x

Scalar	float	double	double- double	quad- double
Solve time (s)	2.6	5.3	29.9	76.5
Accuracy	10 ⁻⁶	10 ⁻¹²	10 ⁻²⁴	10 ⁻⁴⁸

Arbitrary precision solves
using Tpetra and Belos
linear solver package



FP Accuracy Analysis: FloatShadowDouble Datatype

```
class FloatShadowDouble {
```

```
public:
```

```
FloatShadowDouble( ) {
```

```
    f = 0.0f;
```

```
    d = 0.0; }
```

```
FloatShadowDouble( const FloatShadowDouble & fd) {
```

```
    f = fd.f;
```

```
    d = fd.d; }
```

```
...
```

```
inline FloatShadowDouble operator+= (const FloatShadowDouble & fd ) {
```

```
    f += fd.f;
```

```
    d += fd.d;
```

```
    return *this; }
```

```
...
```

```
inline std::ostream& operator<<(std::ostream& os, const FloatShadowDouble& fd) {
```

```
    os << fd.f << "f " << fd.d << "d"; return os;}
```

- Templates enable new analysis capabilities
- Example: Float with “shadow” double.

FloatShadowDouble

Sample usage:

```
#include "FloatShadowDouble.hpp"
```

```
Tpetra::Vector<FloatShadowDouble> x, y;
```

```
Tpetra::CrsMatrix<FloatShadowDouble> A;
```

```
A.apply(x, y); // Single precision, but double results also computed, available
```

Initial Residual =	455.194f	455.194d
Iteration = 15	Residual = 5.07328f	5.07618d
Iteration = 30	Residual = 0.00147022f	0.00138466d
Iteration = 45	Residual = 5.14891e-06f	2.09624e-06d
Iteration = 60	Residual = 4.03386e-09f	7.91927e-10d



Hybrid CPU/GPU Computing

Writing and Launching Heterogeneous Jobs

- A node is a shared-memory domain.
- Multiple nodes are coupled via a communicator.
 - This requires launching **multiple processes**.
- In a heterogeneous cluster, this requires code written for multiple node types.
- It may be necessary to template large parts of the code and run the appropriate instantiation on each rank.
- For launching, two options are available:
 - Multiple single-node executables, complex dispatch
 - One diverse executable, early branch according to rank

Tpetra::HybridPlatform

- Encapsulate main in a templated class method:

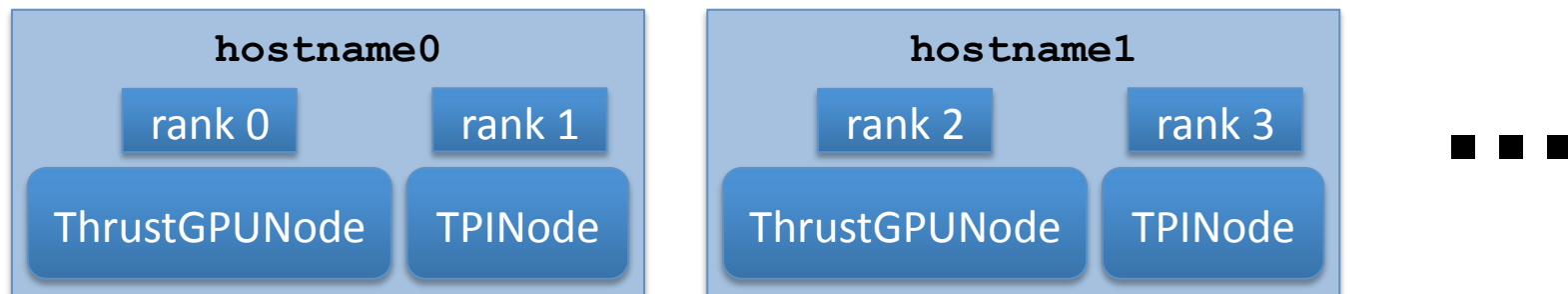
```
template <class Node>
class myMainRoutine {
    static void run(ParameterList &runParams,
                    const RCP<const Comm<int> > &comm,
                    const RCP<Node> &node)
    {
        // do something interesting
    }
};
```

- HybridPlatform maps the communicator rank to the Node type, instantiates a node and the run routine:

```
int main(...) {
    Comm<int>      comm          = ...
    ParameterList machine_file = ...
    // instantiate appropriate node and myMainRoutine
    Tpetra::HybridPlatform platform( comm , machine_file );
    platform.runUserCode< myMainRoutine >();
    return 0;
}
```


HybridPlatform Machine File

round-robin assignment	interval assignment	explicit assignment	default
%M=N	[M,N]	=N	default



```

<ParameterList>
  <ParameterList name="%2=0">
    <Parameter name="NodeType" type="string" value="Kokkos::ThrustGPUNode"/>
    <Parameter name="Verbose" type="int" value="1"/>
    <Parameter name="Device Number" type="int" value="0"/>
    <Parameter name="Node Weight" type="int" value="4"/>
  </ParameterList>
  <ParameterList name="%2=1">
    <Parameter name="NodeType" type="string" value="Kokkos::TPINode"/>
    <Parameter name="Verbose" type="int" value="1"/>
    <Parameter name="Num Threads" type="int" value="15"/>
    <Parameter name="Node Weight" type="int" value="15"/>
  </ParameterList>
</ParameterList>

```

HybridPlatformTest Output

```
[tpetra/example/HybridPlatform] mpirun -np 4 ./Tpetra_HybridPlatformTest.exe  
--machine-file=machines/G+15.xml
```

Every proc machine parameters from: machines/G+15.xml

Teuchos::GlobalMPISession::GlobalMPISession(): started with name **lens31** and rank 0!
Running test with Node == **Kokkos::ThrustGPUNode** on rank 0/4
ThrustGPUNode attached to device #0 "Tesla C1060", of compute capability 1.3

Teuchos::GlobalMPISession::GlobalMPISession(): started with name **lens31** and rank 1!
Running test with Node == **Kokkos::TPINode** on rank 1/4

Teuchos::GlobalMPISession::GlobalMPISession(): started with name **lens10** and rank 2!
Running test with Node == **Kokkos::ThrustGPUNode** on rank 2/4
TPINode initializing with numThreads == 15
ThrustGPUNode attached to device #0 "Tesla C1060", of compute capability 1.3

Teuchos::GlobalMPISession::GlobalMPISession(): started with name **lens10** and rank 3!
Running test with Node == **Kokkos::TPINode** on rank 3/4
TPINode initializing with numThreads == 15

...

See **HybridPlatformAnasazi.cpp** and **HybridPlatformBelos.cpp** for more fun!



Programming Today for Tomorrow's Machines



Programming Today for Tomorrow's Machines

- Parallel Programming in the small:
 - Focus: writing sequential code fragments.
 - Programmer skills:
 - 10%: Pattern/framework experts (domain-aware).
 - 90%: Domain experts (pattern-aware)
- Languages needed are already here.
 - Exception: Large-scale data-intensive graph?

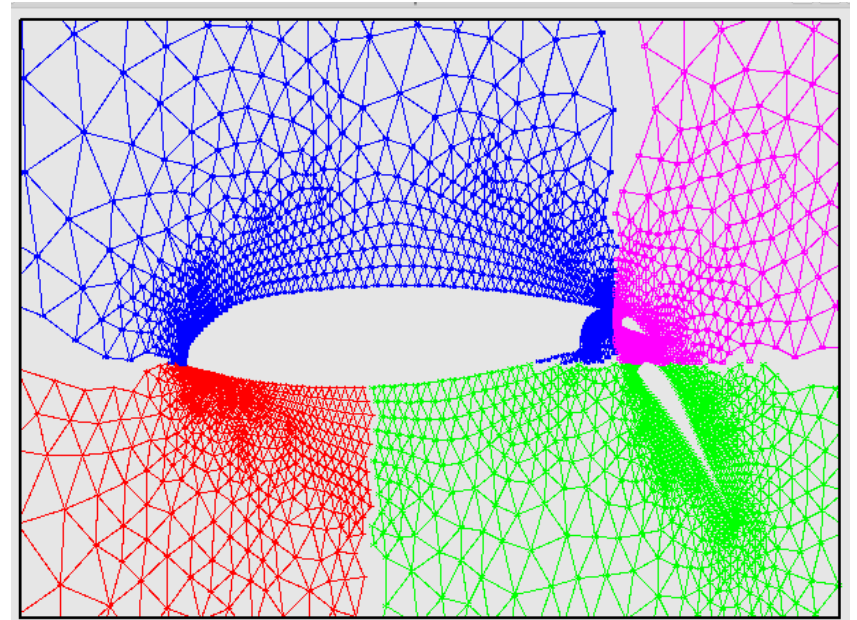


FE/FV/FD Parallel Programming Today

```
for ((i,j,k) in points/elements on subdomain) {  
    compute coefficients for point (i,j,k)  
    inject into global matrix  
}
```

Notes:

- User in charge of:
 - Writing physics code.
 - Iteration space traversal.
 - Storage association.
- Pattern/framework/runtime in charge of:
 - SPMD execution.



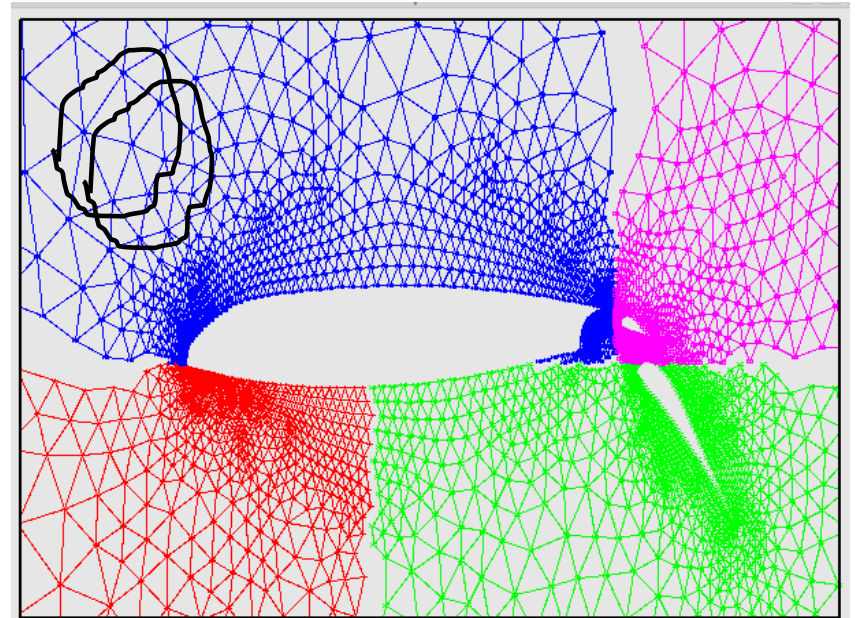


FE/FV/FD Parallel Programming Tomorrow

```
pipeline <i,j,k> {  
    filter(addPhysicsLayer1<i,j,k>);  
    ...  
    filter(addPhysicsLayerN<i,j,k>);  
    filter(injectIntoGlobalMatrix<i,j,k>);  
}
```

Notes:

- User in charge of:
 - Writing physics code (filter).
 - Registering filter with framework.
- Pattern/framework/runtime in charge of:
 - SPMD execution.
 - Iteration space traversal.
 - Sensitive to temporal locality.
 - Filter execution scheduling.
 - Storage association.
- Better assignment of responsibility (in general).





Resilient Algorithms



My Luxury in Life (wrt FT/Resilience)

The privilege to think of a computer as a *reliable, digital* machine.

“At 8 nm process technology, it will be harder to tell a 1 from a 0.”

(W. Camp)



Users' View of the System Now

- “All nodes up and running.”
- Certainly nodes fail, but invisible to user.
- No need for me to be concerned.
- Someone else's problem.



Users' View of the System Future

- Nodes in one of four states.
 1. Dead.
 2. Dying (perhaps producing faulty results).
 3. Reviving.
 4. Running properly:
 - a) Fully reliable or...
 - b) Maybe still producing an occasional bad result.



Hard Error Futures

- C/R will continue as dominant approach:
 - Global state to global file system OK for small systems.
 - Large systems: State control will be localized, use SSD.
- Checkpoint-less restart:
 - Requires full vertical HW/SW stack co-operation.
 - Very challenging.
 - Stratified research efforts not effective.



Soft Error Futures

- Soft error handling: A legitimate algorithms issue.
- Programming model, runtime environment play role.



Consider GMRES as an example of how soft errors affect correctness

- Basic Steps
 - 1) Compute Krylov subspace (preconditioned sparse matrix-vector multiplies)
 - 2) Compute orthonormal basis for Krylov subspace (matrix factorization)
 - 3) Compute vector yielding minimum residual in subspace (linear least squares)
 - 4) Map to next iterate in the full space
 - 5) Repeat until residual is sufficiently small
- More examples in Bronevetsky & Supinski, 2008



Why GMRES?

- Many apps are implicit.
- Most popular (nonsymmetric) linear solver is preconditioned GMRES.
- Only small subset of calculations need to be reliable.
 - GMRES is iterative, but also direct.



Every calculation matters

Description	Iters	FLOPS	Recursive Residual Error	Solution Error
All Correct Calcs	35	343M	4.6e-15	1.0e-6
Iter=2, $y[1] += 1.0$ SpMV incorrect Ortho subspace	35	343M	6.7e-15	3.7e+3
$Q[1][1] += 1.0$ Non-ortho subspace	N/C	N/A	7.7e-02	5.9e+5

- Small PDE Problem: ILUT/GMRES
- Correct result: 35 Iters, 343M FLOPS
- 2 examples of a **single** bad op.
- **Solvers:**
 - 50-90% of total app operations.
 - Soft errors most likely in solver.
- **Need new algorithms for soft errors:**
 - Well-conditioned wrt errors.
 - **Decay proportional to number of errors.**
 - Minimal impact when no errors.

Soft Error Resilience

- New Programming Model Elements: SW-enabled, highly reliable:
 - Data storage, paths.
 - Compute regions.
- Idea: *New algorithms with minimal usage of high reliability.*
- First new algorithm: Flexible-operator (FO)-GMRES.
 - Resilient to soft errors.
 - Only orthogonalization vectors and computations highly reliable.
 - Vast majority of data, ops done with base reliability:
 - Operator, preconditioner data
 - SpMV, Preconditioner application



Software Development and Delivery



“Are C++ templates safe? No, but they are good.”

Compile-time Polymorphism

Templates and Sanity upon a shifting foundation

Software delivery:

- Essential Activity

How can we:

- Implement mixed precision algorithms?
- Implement generic fine-grain parallelism?
- Support hybrid CPU/GPU computations?
- Support extended precision?
- Explore redundant computations?
- Prepare for both exascale “swim lanes”?

C++ templates only sane way:

- Moving to completely templated Trilinos libraries.
- Other important benefits.
- **A usable stack exists now in Trilinos.**

Template Benefits:

- Compile time polymorphism.
- True generic programming.
- No runtime performance hit.
- Strong typing for mixed precision.
- Support for extended precision.
- Many more...

Template Drawbacks:

- Huge compile-time performance hit:
 - But good use of multicore :)
 - Eliminated for common data types.
- Complex notation:
 - Esp. for Fortran & C programmers).
 - Can insulate to some extent.



Solver Software Stack



Phase I packages: SPMD, int/double

Phase II packages: Templated

Optimization Unconstrained: Constrained:	Find $u \in \mathbb{R}^n$ that minimizes $g(u)$ Find $x \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$ that minimizes $g(x, u)$ s.t. $f(x, u) = 0$	Sensitivities (Automatic Differentiation: Sacado)	MOOCHO
Bifurcation Analysis	Given nonlinear operator $F(x, u) \in \mathbb{R}^{n+m}$ For $F(x, u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$		LOCA
Transient Problems DAEs/ODEs:	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0, T], x(0) = x_0, \dot{x}(0) = x'_0$ for $x(t) \in \mathbb{R}^n, t \in [0, T]$		Rythmos
Nonlinear Problems	Given nonlinear operator $F(x) \in \mathbb{R}^m \rightarrow \mathbb{R}^m$ Solve $F(x) = 0 \quad x \in \mathbb{R}^n$		NOX
Linear Problems Linear Equations: Eigen Problems:	Given Linear Ops (Matrices) $A, B \in \mathbb{R}^{m \times n}$ Solve $Ax = b$ for $x \in \mathbb{R}^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \mathbb{R}^n, \lambda \in \mathbb{C}$		Anasazi Ifpack, ML, etc... AztecOO
Distributed Linear Algebra Matrix/Graph Equations: Vector Problems:	Compute $y = Ax; A = A(G); A \in \mathbb{R}^{m \times n}, G \in \mathbb{S}^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \mathbb{R}^n$		Epetra Teuchos



Solver Software Stack



Phase I packages

Phase II packages

Phase III packages: Manycore*, templated

Optimization Unconstrained: Constrained:	Find $u \in \mathbb{R}^n$ that minimizes $g(u)$ Find $x \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$ that minimizes $g(x, u)$ s.t. $f(x, u) = 0$	Sensitivities (Automatic Differentiation: Sacado)	MOOCHO	
Bifurcation Analysis	Given nonlinear operator $F(x, u) \in \mathbb{R}^{n+m}$ For $F(x, u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$		LOCA	T-LOCA
Transient Problems DAEs/ODEs:	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0, T], x(0) = x_0, \dot{x}(0) = x_0'$ for $x(t) \in \mathbb{R}^n, t \in [0, T]$		Rythmos	
Nonlinear Problems	Given nonlinear operator $F(x) \in \mathbb{R}^m \rightarrow \mathbb{R}^m$ Solve $F(x) = 0 \quad x \in \mathbb{R}^n$		NOX	T-NOX
Linear Problems Linear Equations: Eigen Problems:	Given Linear Ops (Matrices) $A, B \in \mathbb{R}^{m \times n}$ Solve $Ax = b$ for $x \in \mathbb{R}^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \mathbb{R}^n, \lambda \in \mathbb{C}$		Anasazi	
Distributed Linear Algebra Matrix/Graph Equations: Vector Problems:	Compute $y = Ax; A = A(G); A \in \mathbb{R}^{m \times n}, G \in \mathfrak{S}^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \mathbb{R}^n$		AztecOO Ifpack, ML, etc...	Belos* T-Ifpack*, T-ML*, etc...
			Epetra	Tpetra* Kokkos*
		Teuchos		

75



Summary

- Some app targets will change:
 - Advanced modeling and simulation: Gives a better answer.
 - Kernel set changes (including redundant computation).
- Resilience requires an integrated strategy:
 - Most effort at the system/runtime level.
 - C/R (with localization) will continue at the app level.
 - Resilient algorithms will mitigate soft error impact.
 - Use of validation in solution hierarchy can help.
- Building the next generation of parallel applications requires enabling domain scientists:
 - Write sophisticated methods.
 - Do so with serial fragments.
 - Fragments hoisted into scalable, resilient fragment.
- Success of manycore will require breaking out of global SIMT-only.
- Migration of Fortran apps to manycore will be painful.



Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have `MPI_Init()`.
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
5. DRY is not possible across CPUs and GPUs
6. Extended precision is too expensive to be useful.
7. Resilience will be built into algorithms.
8. GPUs are a harbinger of CPU things to come.
9. Fortran Developers are in trouble in a manycore world.
10. Global SIMT is sufficient parallelism for scientific computing.