# SOS-11 Panel on Petascale (or is it Exascale?) Systems

**Steve Scott**
**Cray CTO**

SOS-11
Key West, Florida
June 12, 2007

# Questions from Steve P.:

Challenges/technology elements/walls: heterogeneous multicore, accelerators, memory wall, local memory capacity, memory technology, network technology, etc.

(1) How can we alleviate any of these issues?
(2) Are there software solutions on the horizon that will help utilize any of these technologies?
(3) Which of these walls most limits realizing sustained petascale performance?
(4) What technologies could be funded to make the biggest impact on peta/exascale computing?
(5) How long before heterogeneous multicore chips are the norm?
(6) What new processor technologies might be useful/available by 2015?
(7) Acceleration technologies to help current MPI libraries? Will we move to hybrid programming models? What cooperative partnerships could be formed to accelerate the necessary technologies for exascale < 2020.
(8) What will the problems be to reach a usable exdaflop by 2015? Which solutions will get us there?
(9) What are the truly transformational research direct we need to pursue?
(10) What can we do to get back to > 4 B/s per flop and scatter/gather? Just 3D stacking?
(11) What processor interconnects might be available to increase bandwidth between processors? Optical?

# Things I Don't Worry Too Much About

- Scaling the system architecture to 100,000 sockets
  - High-bandwidth networks with load balancing and adaptive routing
  - Globally addressable memory
  - Scalable addressing architecture and translation mechanisms
  - Latency hiding and communication mechanisms
  - Synchronization and collective operation support
- Scaling the system software to 100,000 sockets
  - System boot and administration
  - Basic system services
  - Application launch, scheduling and interaction
- Programming environments to get us to a petascale
  - Programming models:
    - MPI, OpenMP, PGAS, Global View
  - Debugging very challenging at scale; getting by with printf()
  - Performance tuning challenging; lots of time and expertise
  - (Programming environments are being sorely stressed…)

# Key HW Challenges for Peta(Exa)scale Computing

- **On-chip micro-architecture**
  - Move from fast serial thread emphasis to efficient parallel performance
  - Managing data movement and exploiting locality are key
  - *Hardware + Software must isolate the programmer from this!*
- **Local Memory Bandwidth**
  - Absolutely must get beyond DIMMs to 3D integration
  - Possibly nanotube-based memory
- Optical signaling
  - Cable bulk becoming a major issue, and freq*length hitting a wall
  - But optics needs improvement in density, cost, signaling rate
- Power and cooling
  - We can deal with this by limiting performance as needed
  - Lots of headroom for improvement in power management
  - Cooling will likely move back to liquid
- Exotic technologies (cryo, bio, quantum, nano, reversable, etc.)?
  - Not needed for Exascale, with possible exception of nanotube memory
  - Will likely need something beyond CMOS for Zettascale…

# Key SW Challenges for Peta(Exa)scale Computing

- System and application resilience
  - Must ride through all manner of faults
  - Need innovative ways of ensuring that large, long-running apps finish
- File system and parallel I/O scaling
  - Disk bandwidth not scaling (~25%/year/disk vs. ~%100 flops/year)
  - Jitter/load balance a major problem for parallel I/O across many RAIDs
  - Managing data/metadata/authentication/recovery/etc. at petascale
  - ⇒ May be time to insert a new SSD level in the hierarchy
- How do we test large systems we can't deploy in house?
  - Software functionality, reliability, performance
- Better tools for system administrators at scale
  - It's hard to know what's going on in really big systems
- Application scaling
  - Looks like many apps will scale to petaflops; exaflops??
  - Don't want to expose millions of threads to the users
    - ⇒ accelerator technology with compiler-managed node-level parallelism
- Programming languages and tools….
  - Really need a fundamental advancement here to (a) scale to an exaflop and (b) make parallel programming easier for the masses

# What Parallel Programming Model Are We Going to Inflict on the Masses?

Relative complexity of three programming models (as measured by the size of their documentation):

- MPI 1 Standard: 239
  MPI 2 Standard: 376
  Total: **615** Pages

- "intro_shmem" man page: **10** Pages
  (Cray implemented the entire NPB suite with a shmem_get and a barrier)

- CF90 Co-Array Programming Manual (SR-3908): **30** Pages
  (11 pages for language description)

# Random Gather (Serial)

```
parameter (n=2**30)
real table(n)
buffer(nelts)  ! nelts << n
…
do i=1,nelts

   buffer(i) = table(index(i))

enddo
```

- Purely synthetic, but simulates “irregular” communication access patterns.
- (We do have customers that do this stuff)

# Random Gather (Co-Array Fortran)

```
      parameter (n=2**30)                ! 1 Gigaword
      parameter (NPES=2**7)              ! 128 Pes
      parameter (eltspe = 2**23)         ! Elements per PE
      real table (eltspe)[NPES]
…
!dir$ unroll(16)
      do i=1,nelts
        PE =( index(i) + eltspe-1)/eltspe
        offset = mod(index(i)-1,eltspe)+1
        buffer(i) = table(offset)[PE]
      enddo
```

- Simple to write
- Very high performance
- Need the right underlying hardware

# Random Gather (MPI)

```
 if(my_rank.eq.0)then
! first gather indices to send out to individual PEs
        do i=1,nelts
          indpe = ceiling(real(index(i))/real(myelts)) - 1
          isum(indpe)=isum(indpe)+1
          who(isum(indpe),indpe) = index(i)
        enddo
! send out count and indices to PEs
        do i = 1, npes-1
           call MPI_SEND(isum(i),8,MPI_BYTE,i,10,
    &           MPI_COMM_WORLD,ier)
          if(isum(i).gt.0)then
           call MPI_SEND(who(1,i),8*isum(i),MPI_BYTE,i,11,
    &           MPI_COMM_WORLD,ier)
          endif
        enddo
! now wait to receive values and scatter them.
        do i = 1,isum(0)
          offset = mod(who(1,0)-1,myelts)+1
          buff(i,0) = table(offset)
        enddo
        do i = 1,npes-1
          if(isum(i).gt.0)then
           call MPI_RECV(buff(1,i),8*isum(i),MPI_BYTE,i,12,
    &           MPI_COMM_WORLD,status,ier)
          endif
        enddo
        do i=nelts,1,-1
          indpe = ceiling(real(index(i))/real(myelts)) - 1
          offset = isum(indpe)
          isum(indpe) = isum(indpe) - 1
          buffer(i) = buff(offset,indpe)
        enddo
        else  !if my_rank.ne.0
          call MPI_RECV(my_sum,8,MPI_BYTE,0,10,
    &           MPI_COMM_WORLD,status,ier)
          if(my_sum.gt.0)then
           call MPI_RECV(index,8*my_sum,MPI_BYTE,0,11,
    &           MPI_COMM_WORLD,status,ier)
           do i = 1, my_sum
             offset = mod(index(i)-1,myelts)+1
          do i = 1, my_sum
             offset = mod(index(i)-1,myelts)+1
             buffer(i) = table(offset)
           enddo
           call MPI_SEND(buffer,8*my_sum,MPI_BYTE,0,12,
    &           MPI_COMM_WORLD,ier)
          endif
        endif
```

# Chapel

## *A new parallel language developed by Cray for HPCS*

### Themes

- Raise level of abstraction, generality compared to SPMD approaches
- Support prototyping of parallel codes + evolution to production-grade
- Narrow gap between parallel and mainstream languages

### Chapel's Productivity Goals

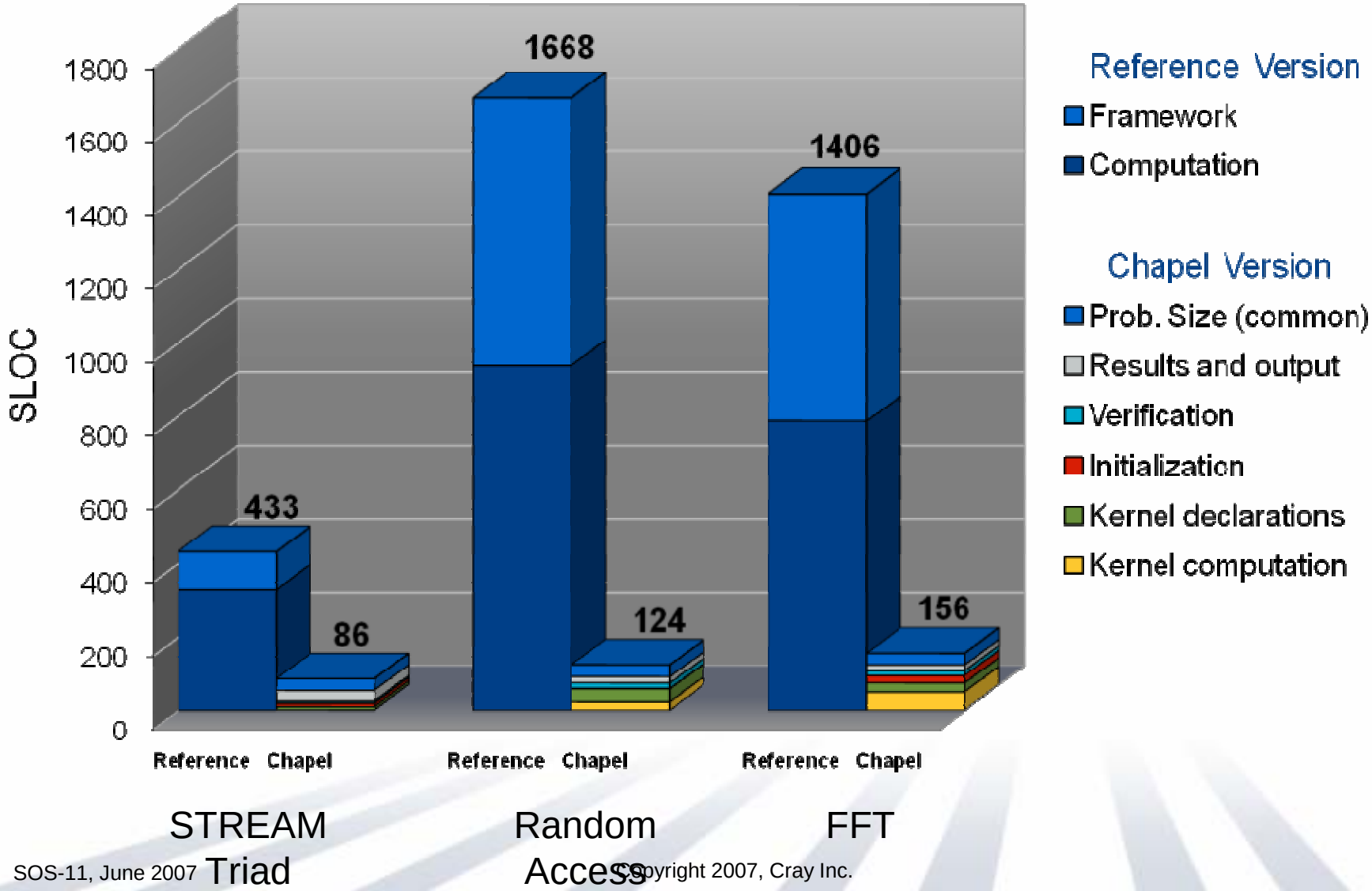
- Vastly improve **programmability** over current languages/models
- Support **performance** that matches or beats MPI
- Improve **portability** over current languages/models (similar to MPI)
- Improve code **robustness** via improved abstractions and semantics

```
...
forall block in
  for r in RAStr
                                  k.low )
                                  r;
```

### Status

- Draft language specification available
- Portable prototype implementation underway
- Performing application kernel studies to exercise Chapel
- Working with HPLS team to evaluate Chapel
- Initial release made to HPLS team in December 2006

# Chapel Code Size Comparison
## For HPC Challenge Benchmarks



# End