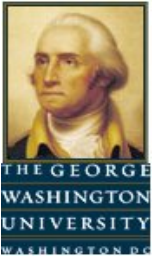


Random Access in UPC

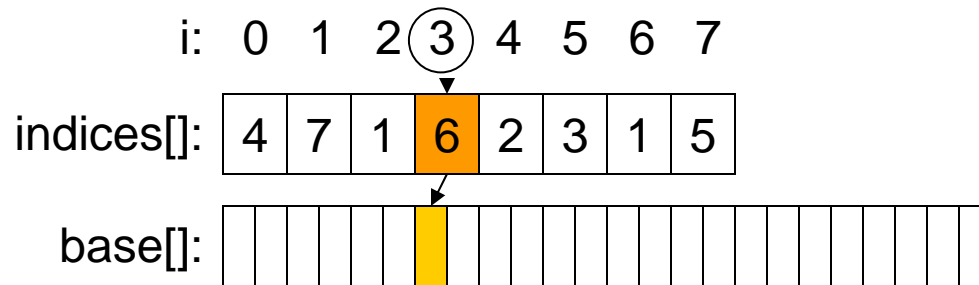
François Cantonnet, Yiyi Yao, Tarek El-Ghazawi

High Performance Computing Laboratory
The George Washington University
September 21st 2004



Giga Updates per Second (GUPS)

$\forall i \in [0 \dots \text{UPDATES}-1], \text{field}[\text{indices}[i]] += 1$



Parameters:

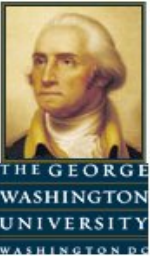
indices = array holding indices into base

updates = size of indices

base = data set array

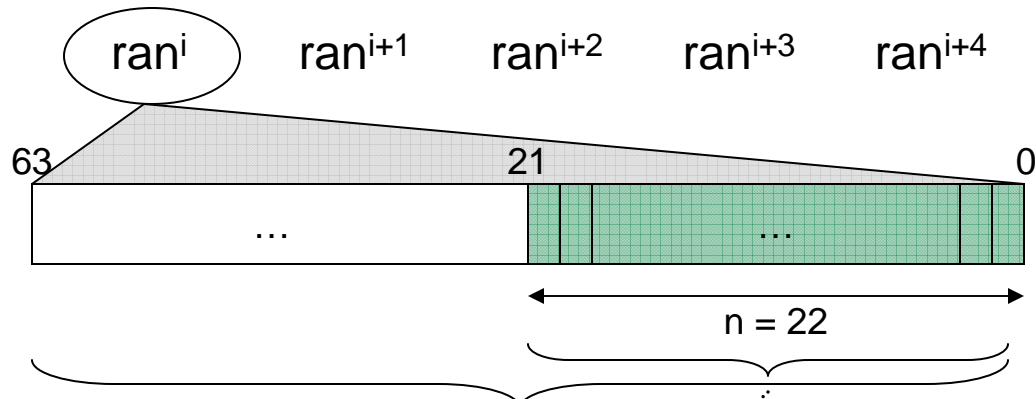
size = size of base

repeats = number of repeats of the whole thing



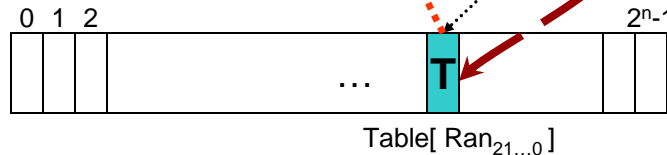
RandomAccess

Ran interpreted as follows:



Sequence of 64-bit random numbers as described by Eq1 of length UPDATES = 2^{n+2}

Table []:



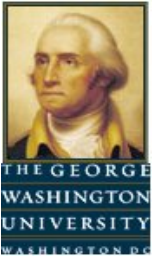
XOR

Objects:

Table = data set array of size 2^n 64-bit integers

Operation: Update of T
 $T = T \text{ XOR } \text{RAN}$

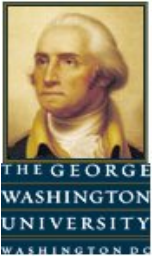
Eq1: $\text{ran}^0 = 1$
 $\text{ran}^n = 2 * \text{ran}^{n-1} \text{ XOR } [\text{ran}^{n-1}_{63} * 7]$



GUPS vs RandomAccess

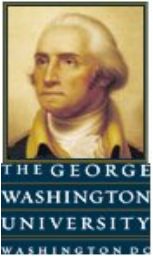
	Type of Computation	Randomization
GUPS	Simple Increment	Precomputed using random() and stored into a table
RandomAccess HPCC v0.5 compliant	XOR Stable[MSB _{1..9} (ran _i)]	Generated on-the-fly by: ran ₀ = 1 ran _n = 2*ran _{n-1} XOR [MSB ₁ (ran _{n-1})*7]
RandomAccess HPCC v0.6 compliant	XOR ran _i	Generated on-the-fly by: ran ₀ = 1 ran _n = 2*ran _{n-1} XOR [MSB ₁ (ran _{n-1})*7]

Note: The number of UPDATES is set to 4 times the problem size



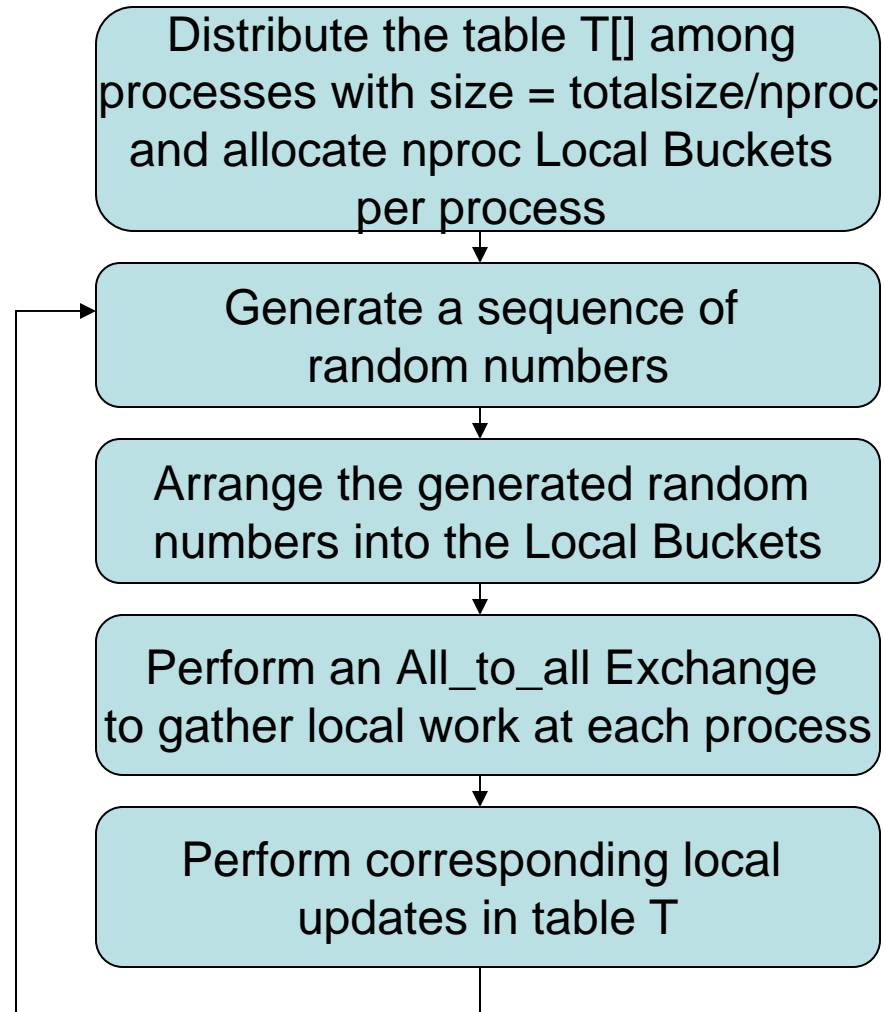
Available codes for RandomAccess

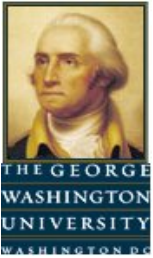
- ◆ The following ‘variants’ of the HPCC RandomAccess codes are available:
 - MPI/1 bulk (MPI/1 using buckets, present in HPCC v0.5alpha, adapted to v0.6 by GWU)
 - MPI/1 using Send/iRecv (from HPCC v0.6)
 - MPI/1 using iSend/iRecv (from HPCC v0.6)
 - MPI/2 (from GWU)
 - UPC (from GWU)



MPI/1 (bulk)

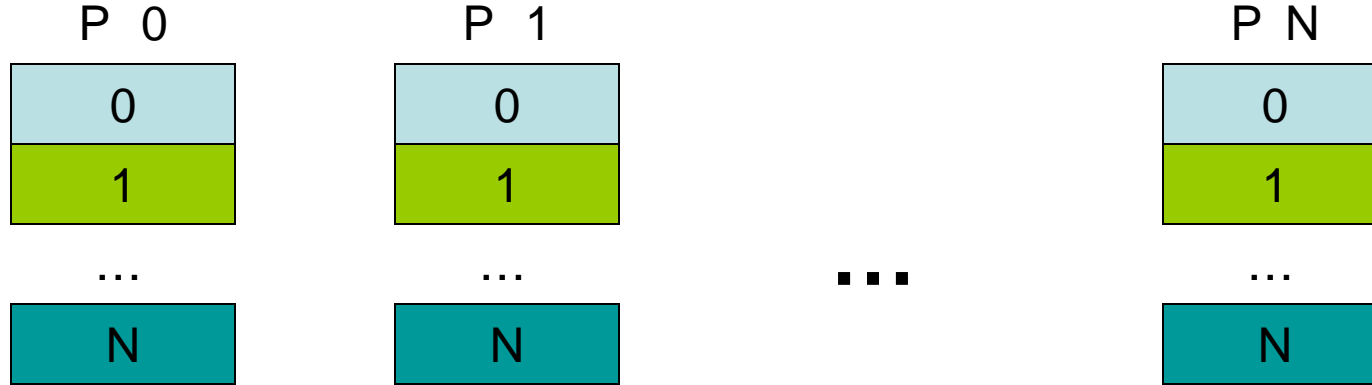
RandomAccess Process





MPI/1 (bulk) RandomAccess Local/Global Bucket Exchange

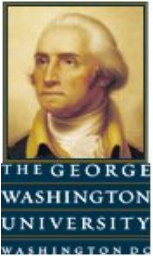
Local Bucket



MPI All-to-All Communication

Global Bucket

0 1 N



MPI/1 (new version)

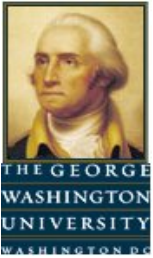
RandomAccess

```
For each update i do
  generate rani
  If update i is local then
    update local table
  Else
    send rani to PEj (UPDATE tag)
  End if
  Process all pending incoming
  UPDATE messages and update
  local table
Done
Send DONE tag to all PEs (except
myself)
Process all incoming UPDATE
messages and update accordingly
the local table until all PEs are
DONE
```

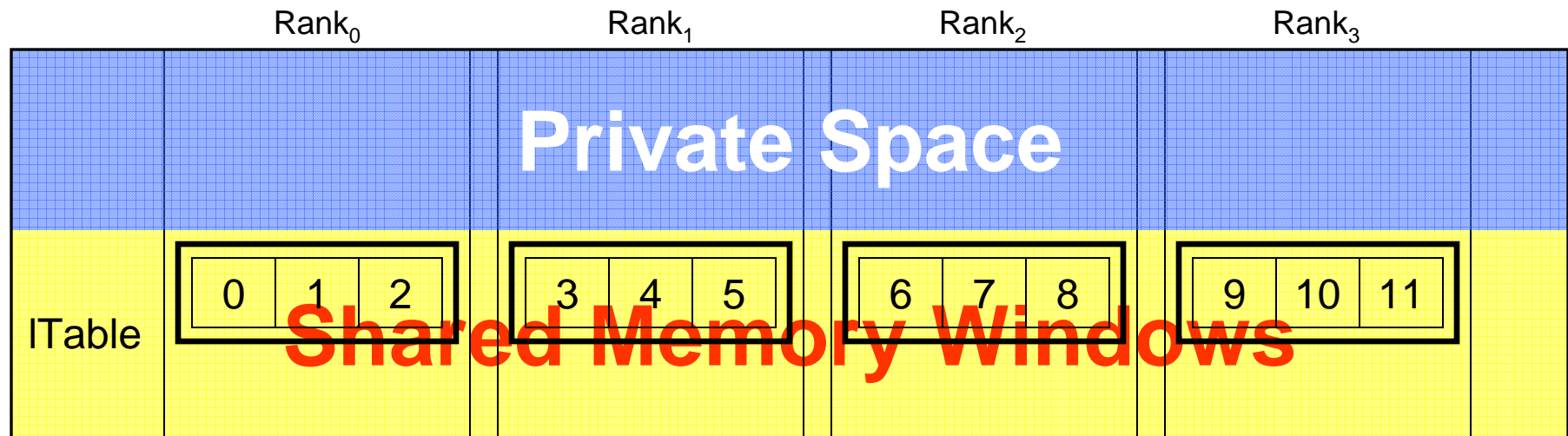
MPI Send/iRecv

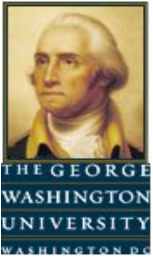
```
For each update i do
  If no send pending then
    generate rani
    If update i is local then
      update local table
    Else
      send rani to PEj (UPDATE tag)
    End if
  End if
  Process all pending incoming
  UPDATE messages and update
  local table
Done
Send DONE tag to all PEs (except
myself)
Process all incoming UPDATE
messages and update accordingly
the local table until all PEs are
DONE
```

MPI iSend/iRecv

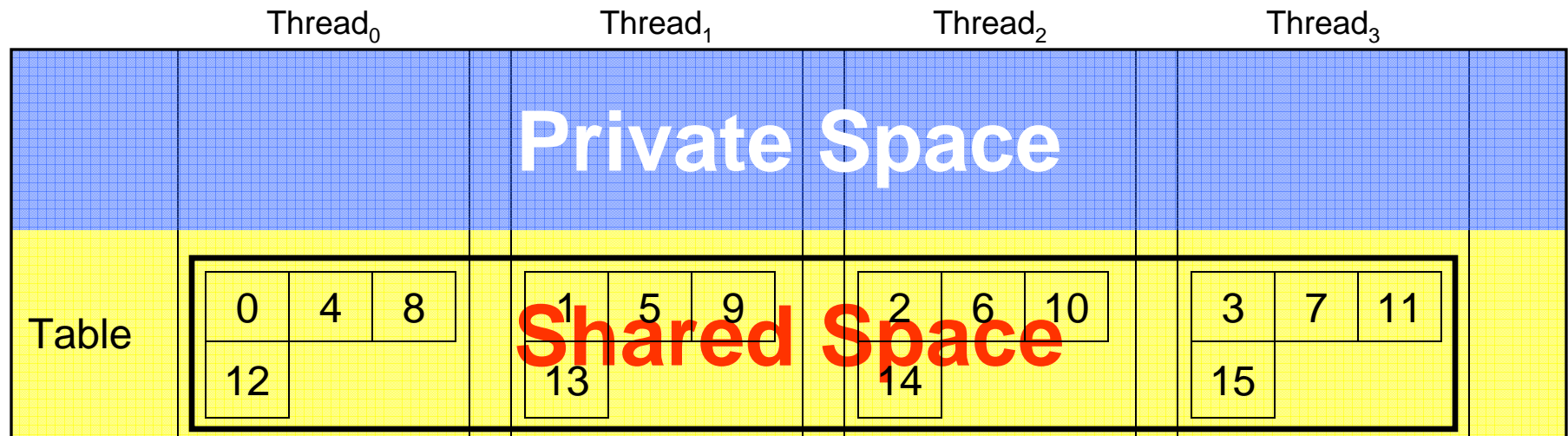


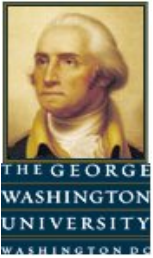
RandomAccess – MPI/2





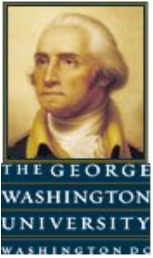
RandomAccess – UPC





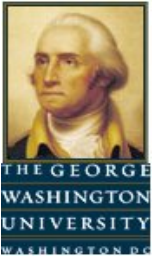
Development Complexity

- ◆ Code size
 - LOC
 - COC
- ◆ Programming models can add conceptual complexity, which affects programming time
 - Domain decomposition
 - Library based programming paradigms versus parallel languages
 - Conceptual efforts are hard to measure
 - ◆ Keywords?
 - ◆ Function calls?
 - ◆ Tokens?
 - ◆ Loop depths?
 - ...



Lines of Codes or Number of Characters?

- ◆ Typical line length varies from one language to the other
- ◆ Conciseness is good
 - TTS
 - Maintainability
- ◆ What matters is how many keystrokes the programmer does, not how many <enters>
 - `main() { ...}, vs.`
`main()`
`{`
`...`
`}`

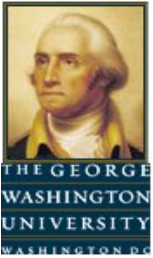


Lines and Character of Code?

	GUPS (#lines)	GUPS (#char)	RandAccess (#lines)	RandAccess (#char)
C	48	1,294	144	3,381
UPC	54	1,423	158	3,628
MPI/2	80	2,382	210	5,776
MPI/1	85	2466	344	9,569

Only the effective number of lines and characters are measured
(excluding comments and blocks declarations '{' and '}')

Study done with HPCC 0.5alpha compliant code



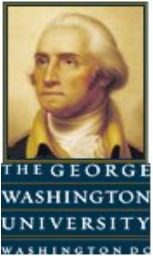
Effort increase

	GUPS (#lines)	GUPS (#char)	RandAccess (#lines)	RandAccess (#char)
(UPC-C)%	12.24%	9.97%	9.72%	7.31%
(MPI/2-C)%	40%	84.08%	45.83%	70.84%
(MPI/1-C)%	77.08%	90.57%	138.89%	183.02%

$$\frac{\textit{Parallel} - \textit{Sequential}}{\textit{Sequential}} \times 100 \%$$

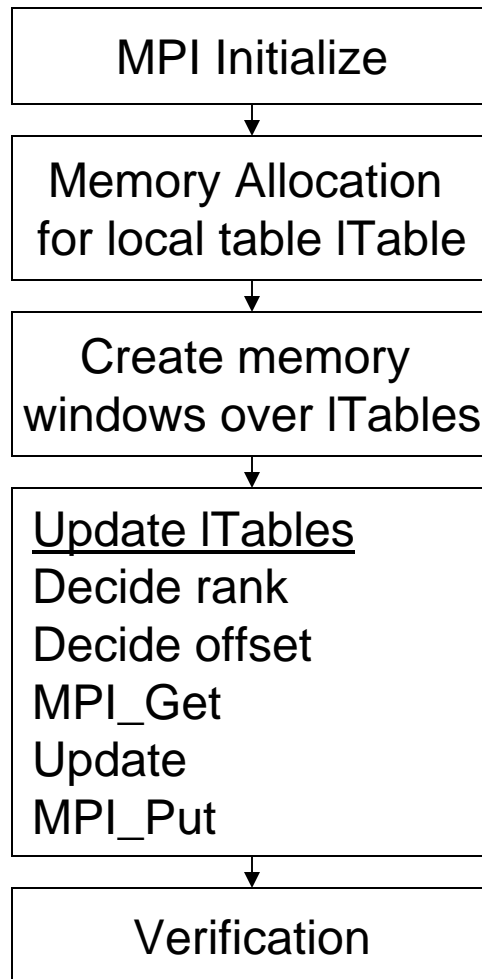
Only the effective number of lines and characters are measured
(excluding comments and blocks declarations '{' and '}')

Study done with HPCC 0.5alpha compliant code

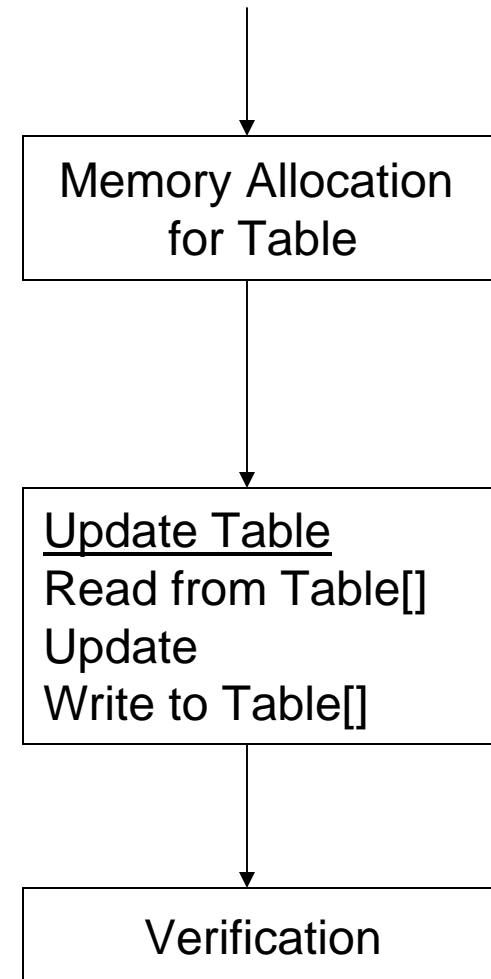


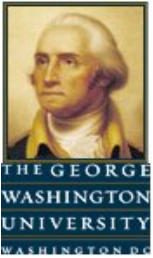
Conceptual Effort MPI/2 vs UPC

MPI/2



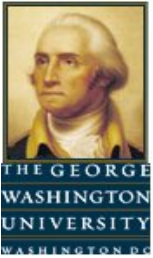
UPC





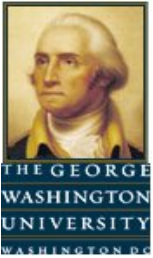
Conceptual Complexity (cont.)

		Work Distr.	Data Distr.	Comm.	Synch. & Consist.	Misc. Ops	Sum	Overall Score
RandomAccess MPI/2	# Parameters	26	9	35	5	6	81	151
	# Function Calls	0	2	8	4	4	18	
	# Keywords with rank and np	15	6	8	0	2	31	
	# MPI Types	0	5	10	4	2	21	
	Notes	11 If 5 For	1 memalloc 1 window create	4 for collective operation 4 one-sided	1 fence 3 barriers (1 implicit)	mpi_init mpi_finalize mpi_comm_rank mpi_comm_size		
RandomAccess UPC	# Parameters	19	2	0	0	2	23	42 ←
	# Function Calls	0	1	0	5	2	8	
	# Keywords	5	1	0	0	0	6	
	# UPC Constructs & UPC Types	3	2	0	0	0	5	
	Notes	3 forall 4 if 1 for	2 shared 1 upc_all_alloc		5 barriers	2 global_exit		



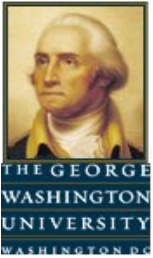
Testbed - RandomAccess

	SGI O2k	Cray X1	HP AlphaServer SC
MPI-1 (bulk)	✓	✓	✓
MPI-1 (Send/iRecv)	✓	✓	✓
MPI-1 (iSend/iRecv)	✓	✓	✓
MPI-2	✓	✓	
GCC-UPC 64b	✓		
Cray X1 UPC		✓	
HP UPC			✓
BUPC (MPI conduit)	✓		
BUPC (SMP conduit + pthreads)	✓		



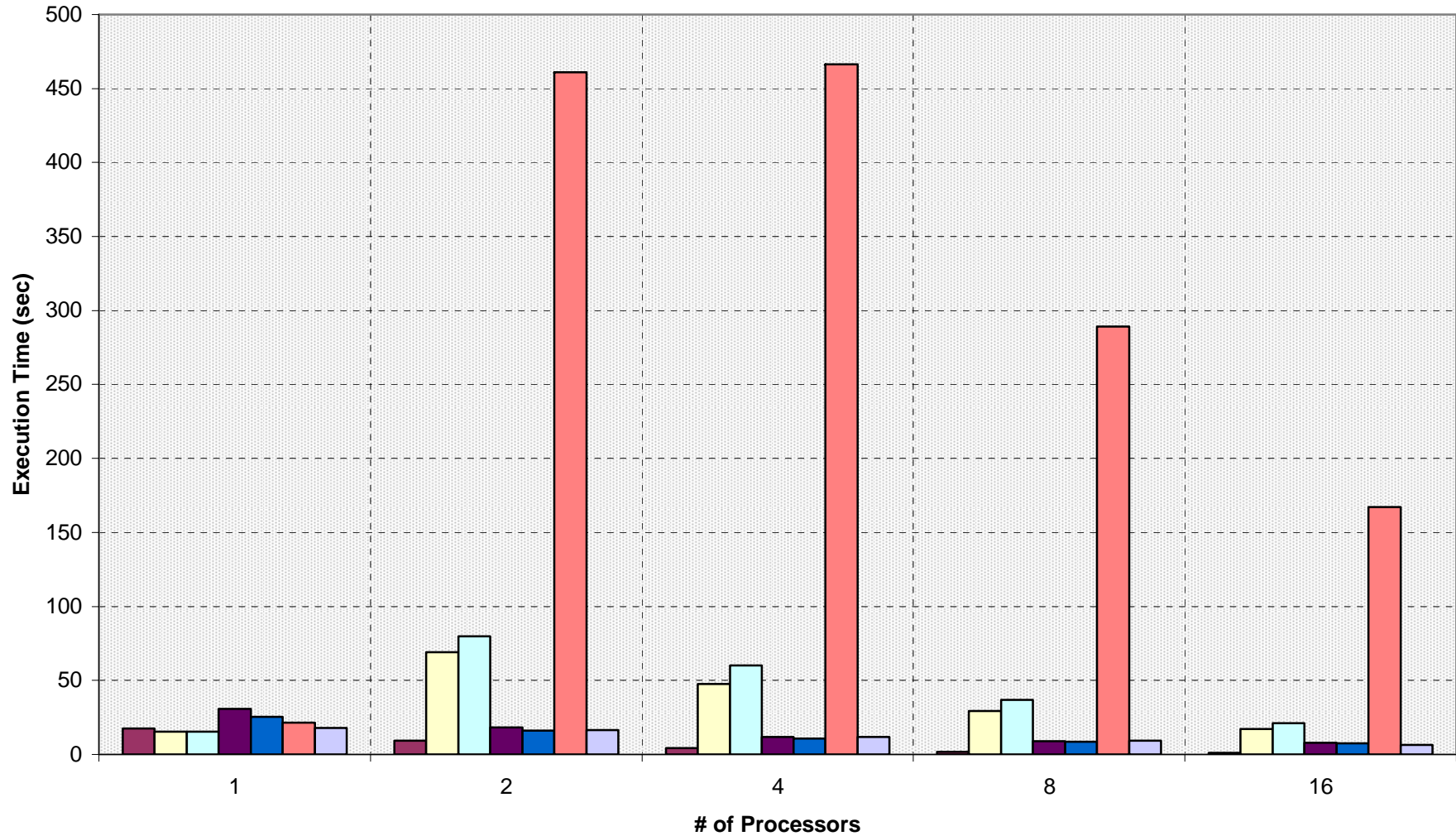
SGI Origin 2000

- ◆ 16 processors, 2 processors per SMP
- ◆ 16 GB of RAM
 - SMP
 - ◆ 2 CPUs
 - MIPS R10000 195MHz IP27
 - » 4MB L2 cache
- ◆ Craylink Interconnect
- ◆ Software:
 - SGI Irix 6.4
 - Intrepid GCC-UPC compiler v3.2.3.5 (64b enabled)
 - Berkeley UPC Compiler v2.0.0 with v1.9.3 runtime system

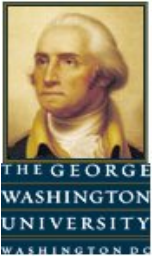


RandomAccess on O2k

RandomAccess v0.6 (4M * 64bit words) - SGI Origin 2000

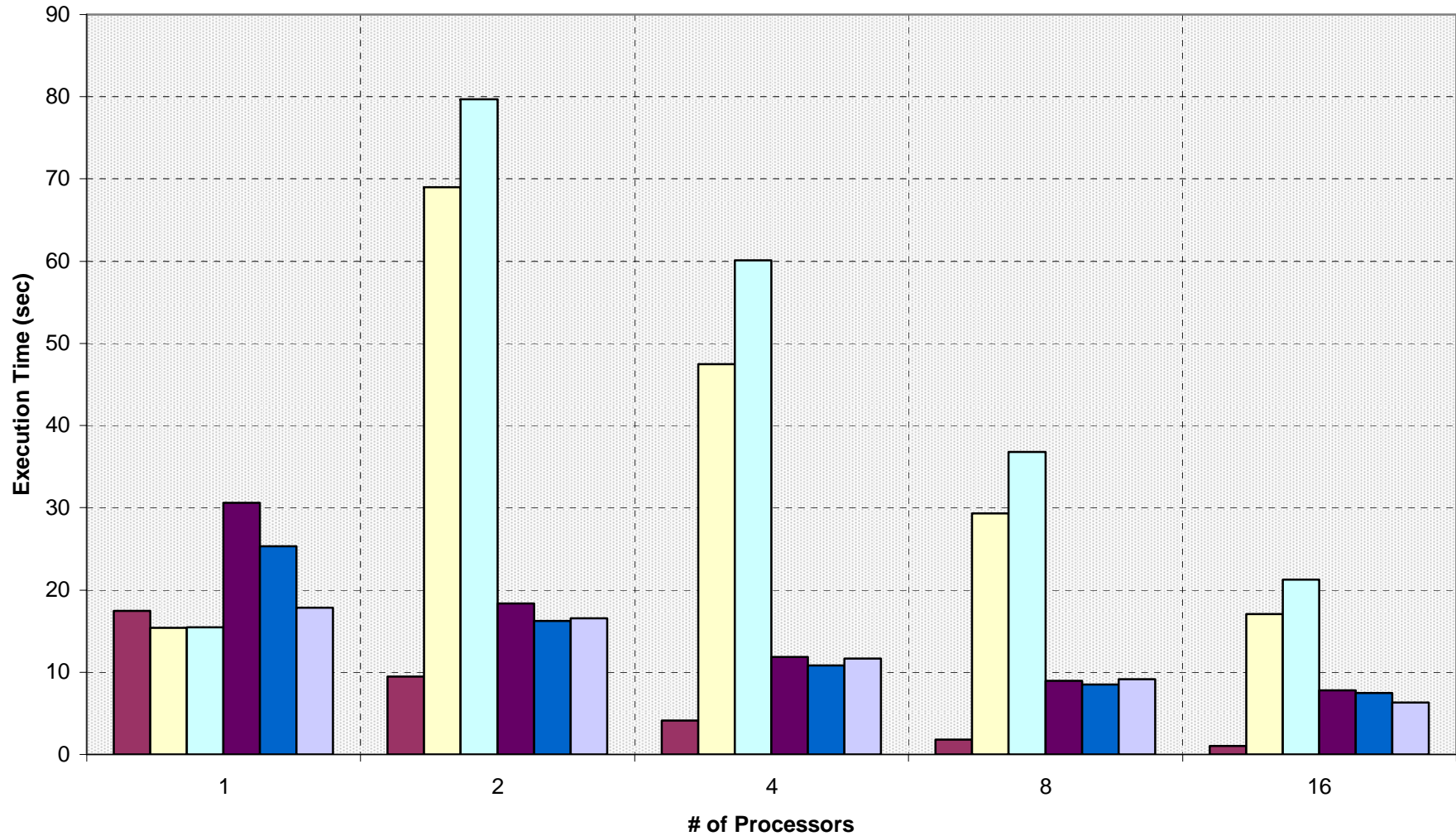


■ MPI old (Bulk) ■ MPI new (Send) ■ MPI new (iSend) ■ MPI-2 ■ GCC-UPC-64 ■ BUPC (MPI conduit) ■ BUPC (SMP conduit with Pthreads)

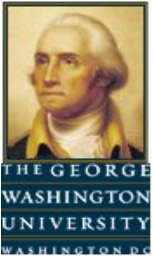


RandomAccess on O2k (cont.)

RandomAccess v0.6 (4M * 64bit words) - SGI Origin 2000

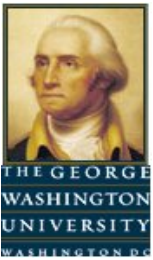


■ MPI old (Bulk) ■ MPI new (Send) ■ MPI new (iSend) ■ MPI-2 ■ GCC-UPC-64 ■ BUPC (SMP conduit with Pthreads)



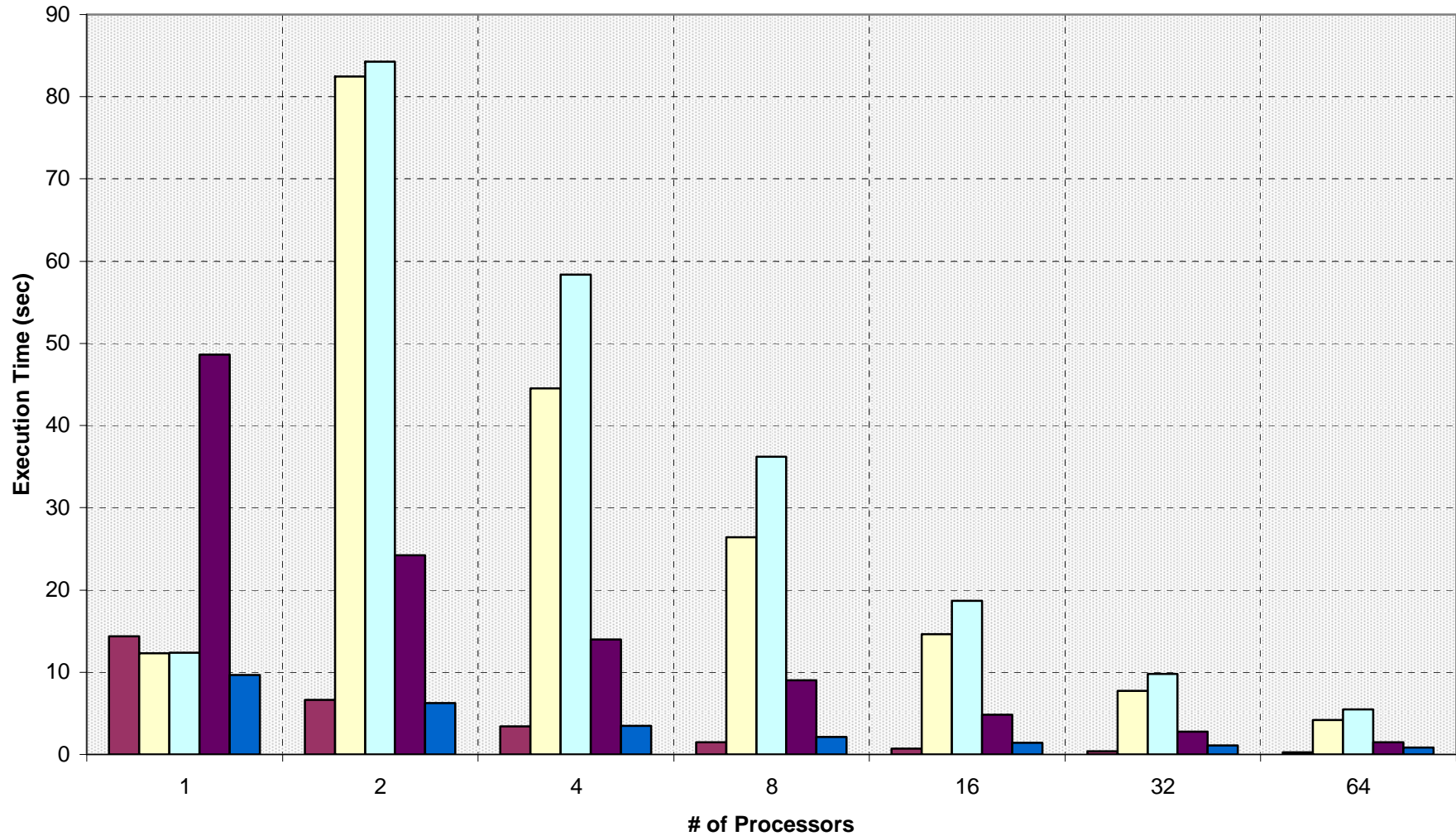
Cray X1

- ◆ 512 Multi-Streaming Vector Processors
 - MSP
 - ◆ 4 SSPs
 - ◆ 2 MB cache
- ◆ 2 TB of RAM
- ◆ Software:
 - PE 5.2
 - Cray X1 UPC compiler

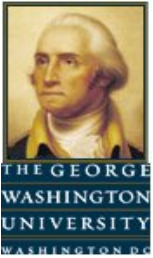


RandomAccess on X1

RandomAccess v0.6 (4M* 64bit words) - Cray X1

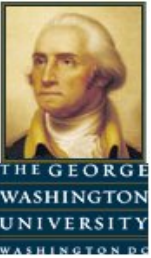


■ MPI old (Bulk) ■ MPI new (Send) ■ MPI new (iSend) ■ MPI-2 ■ UPC



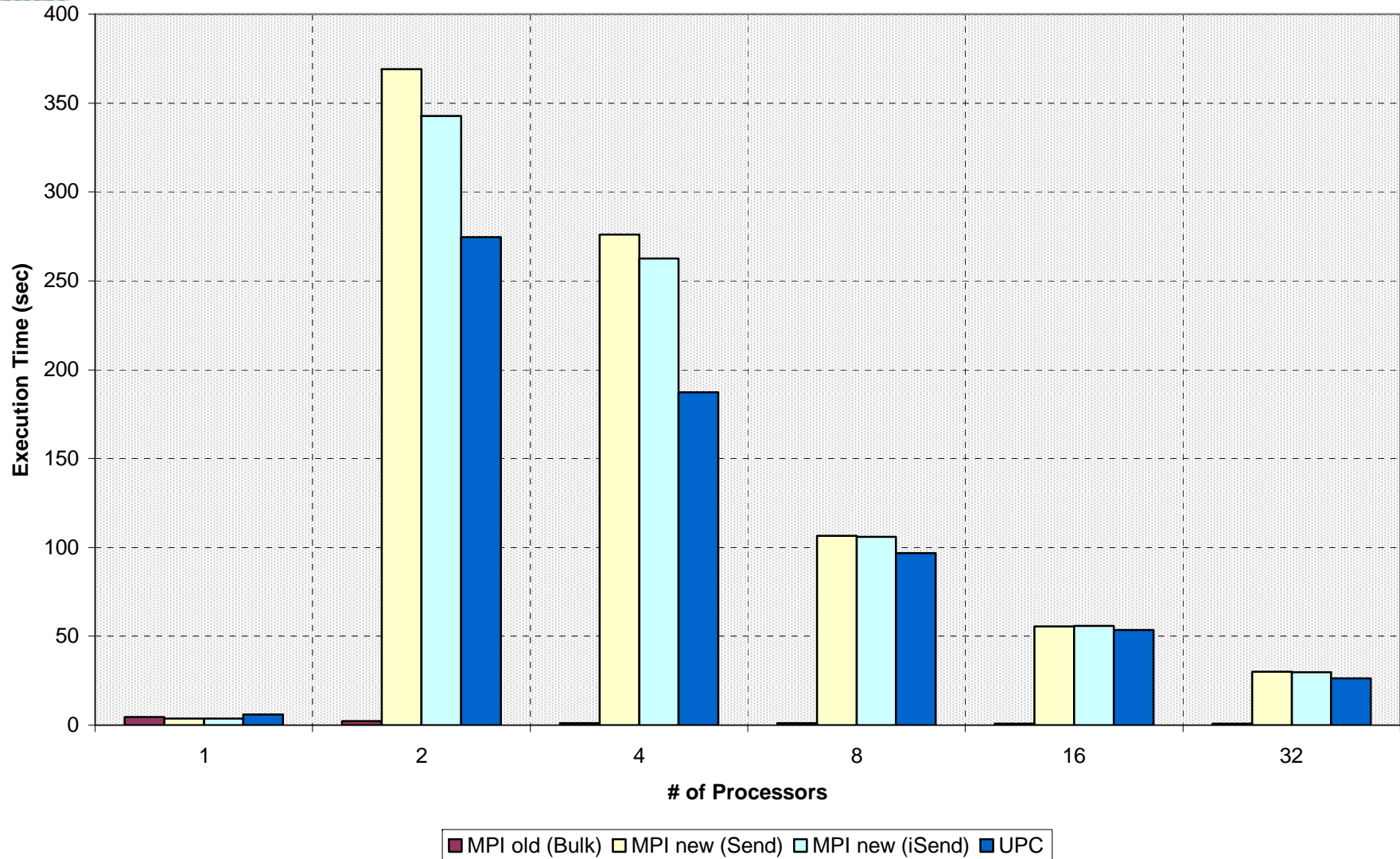
HP AlphaServer SC

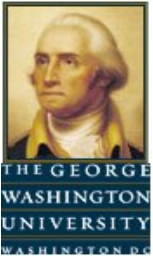
- ◆ Compaq AlphaServer SC V2.5
- ◆ 8 compute nodes:
 - SMP
 - ◆ 4 CPUs
 - Alpha EV6.7
 - » 667MHz,
 - » 8MB L2 cache
 - ◆ 2 GB of RAM
- ◆ Quadrics Interconnect
- ◆ Software:
 - Compaq Tru64 UNIX V5.1A
 - HP UPC Compiler v2.3-002
 - ◆ HP UPC Runtime System v6.0



RandomAccess on HP AlphaServer

RandomAccess v0.6 (4M* 64bit words) - HP Alphaserver SC





Conclusions

- ◆ Programming in UPC seems to be much easier than MPI/2, which in this case was already easier than MPI/1
 - In terms in manual efforts (# lines/#char)
 - ◆ UPC is approx 10% bigger than sequential code
 - ◆ MPI/2 is approx 1.5 to 2 times bigger
 - ◆ MPI/1 is approx 2 to 3 times bigger
 - In terms of conceptual complexity
 - ◆ UPC has a 'score' three times smaller than MPI/2
- ◆ Roughly, as compared to an 'ideal' bulk scheme, UPC is giving:
 - Similar level of performance to 2 times slower on the X1
 - 1.5 to 5 times slower on the O2k
 - 25 to 100 times slower on the AlphaServer (cluster architecture)