Single PE Optimization Techniques for the CRAY T3E System

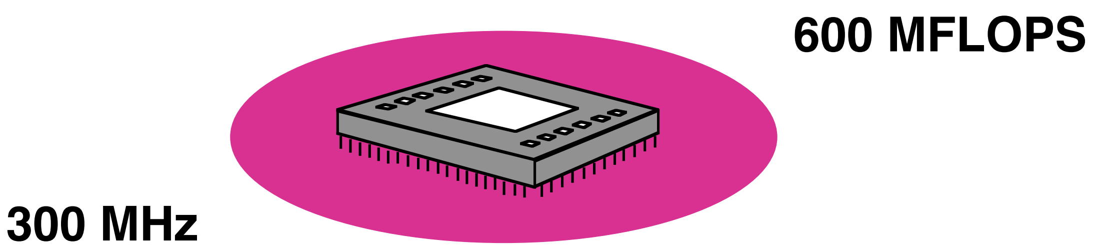
Jeff Brooks, Benchmarking Dept
Cray Research
jpb@cray.com

Outline

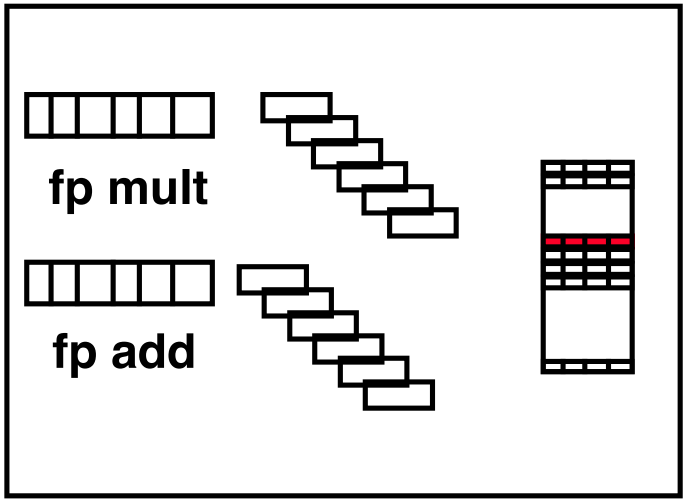
1. Overview of EV5 and implementation in T3E.
   - Cache
   - Memory System & Streams
   - Functional Units
   - E-Registers
   - Cache Coherence
2. Techniques to optimize
   - Unrolling
   - Loop Splitting
   - Benchlib
   - E-Registers
3. Examples

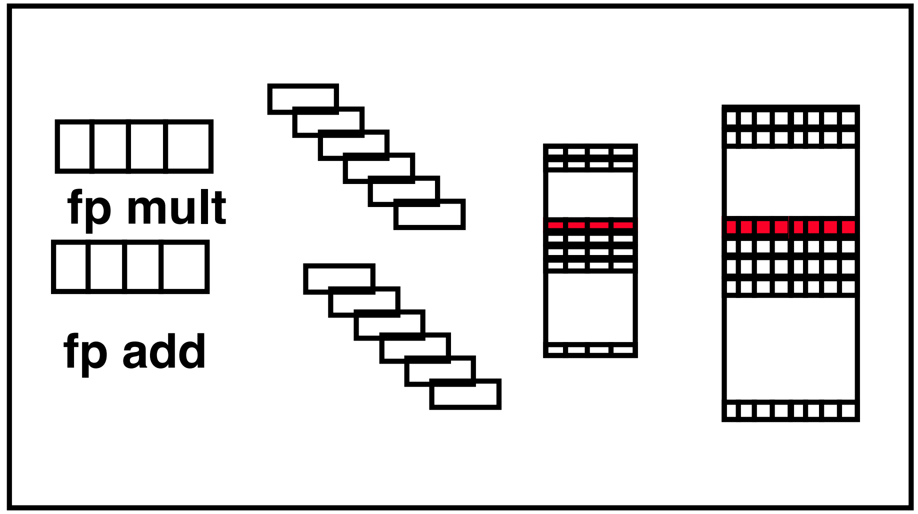
Processing Element (PE) Microarchitecture

- DEC Alpha 21164 microprocessor
  - 300 MHz peak
  - 600 MFLOPS peak
  - On-chip caches
    - 8 KB I cache and 8KB D cache
    - 96 KB secondary D cache (3-way set associative)



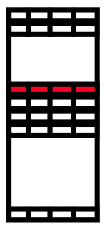
EV4 vs. EV5



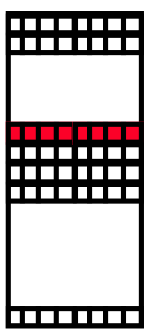


EV4	EV5
32 int and 32 float registers	Same
6 clock fp add/mult pipes (40 ns)	4 clock fp add/mult pipes (13.3 ns)
1K direct mapped cache	Same
No secondary cache	96 KB 3-Way set Associative
dual issue	quad issue
150 Mhz	300 Mhz
	Divide 4X faster

EV5 Cache System



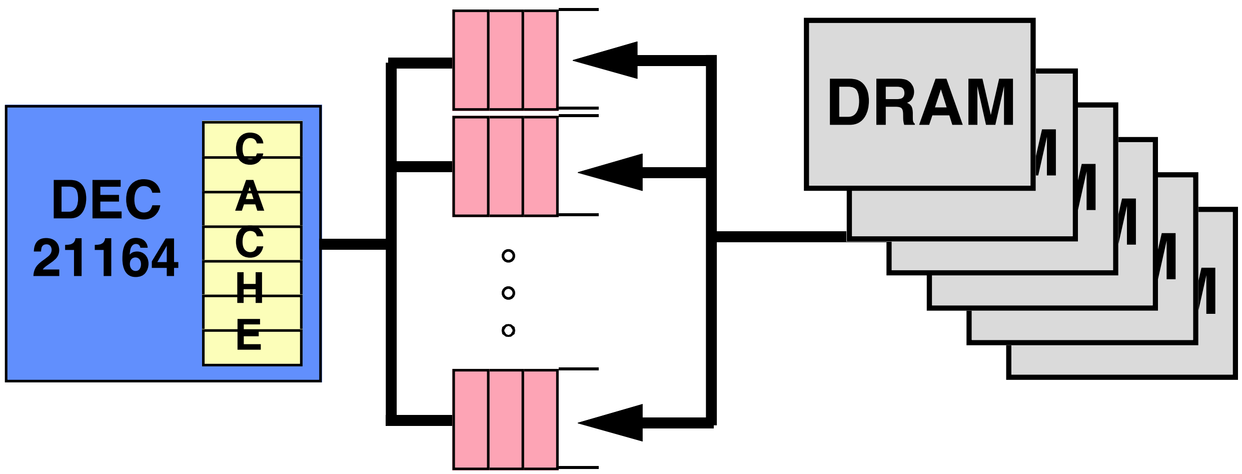
- **Primary Dcache like T3D**
  - 1024 words, 4 word lines
  - direct mapped
  - read allocate, write through
  - On EV5, FP loads frequently bypass this cache!



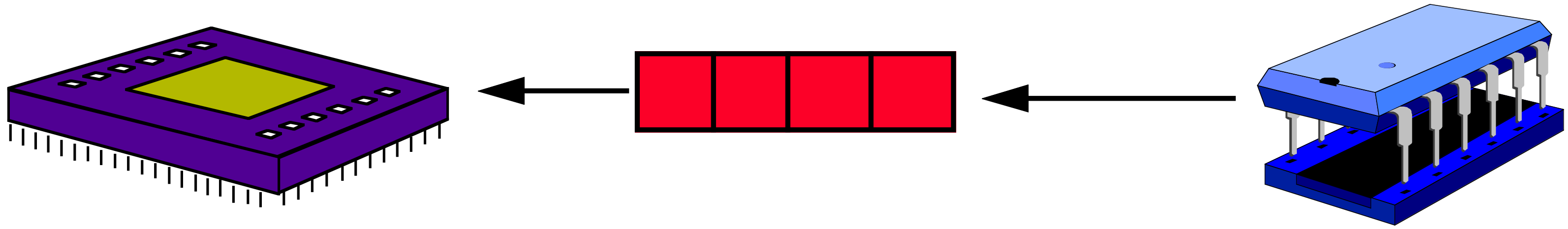
- **On-chip Scache new in EV5**
  - 12288 words, 8 word lines
  - 3-Way set associative with random replacement policy
  - read/write allocate and write back
  - Random replacement implies “best” performance is obtained if you treat it like a 4096 word direct-mapped cache
  - Caches data AND instructions

Local Memory System

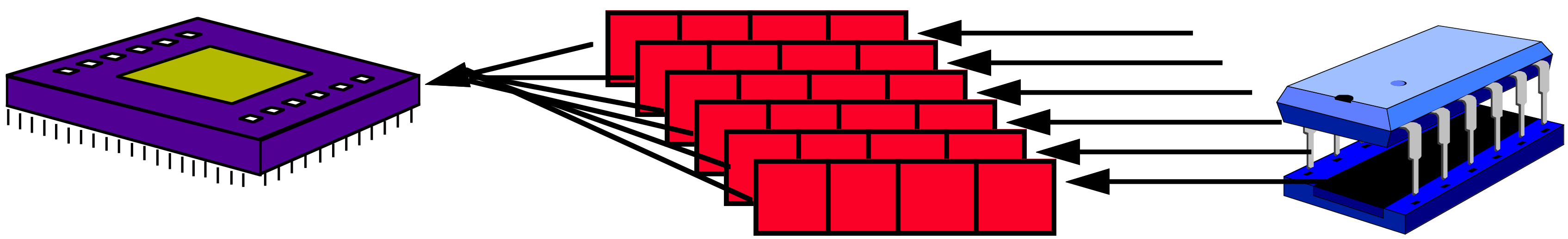
- **Streams (6)**
  - Prefetching from DRAM to fast buffers
  - Automatically hardware managed
  - Stream is allocated after 2 Scache misses
  - Helps take advantage of Page mode of DRAM part
  - Allowed Cray to use EV5 pins for data vs. board level cache design which splits the pins between data and cache signaling.



T3D vs T3E Memory

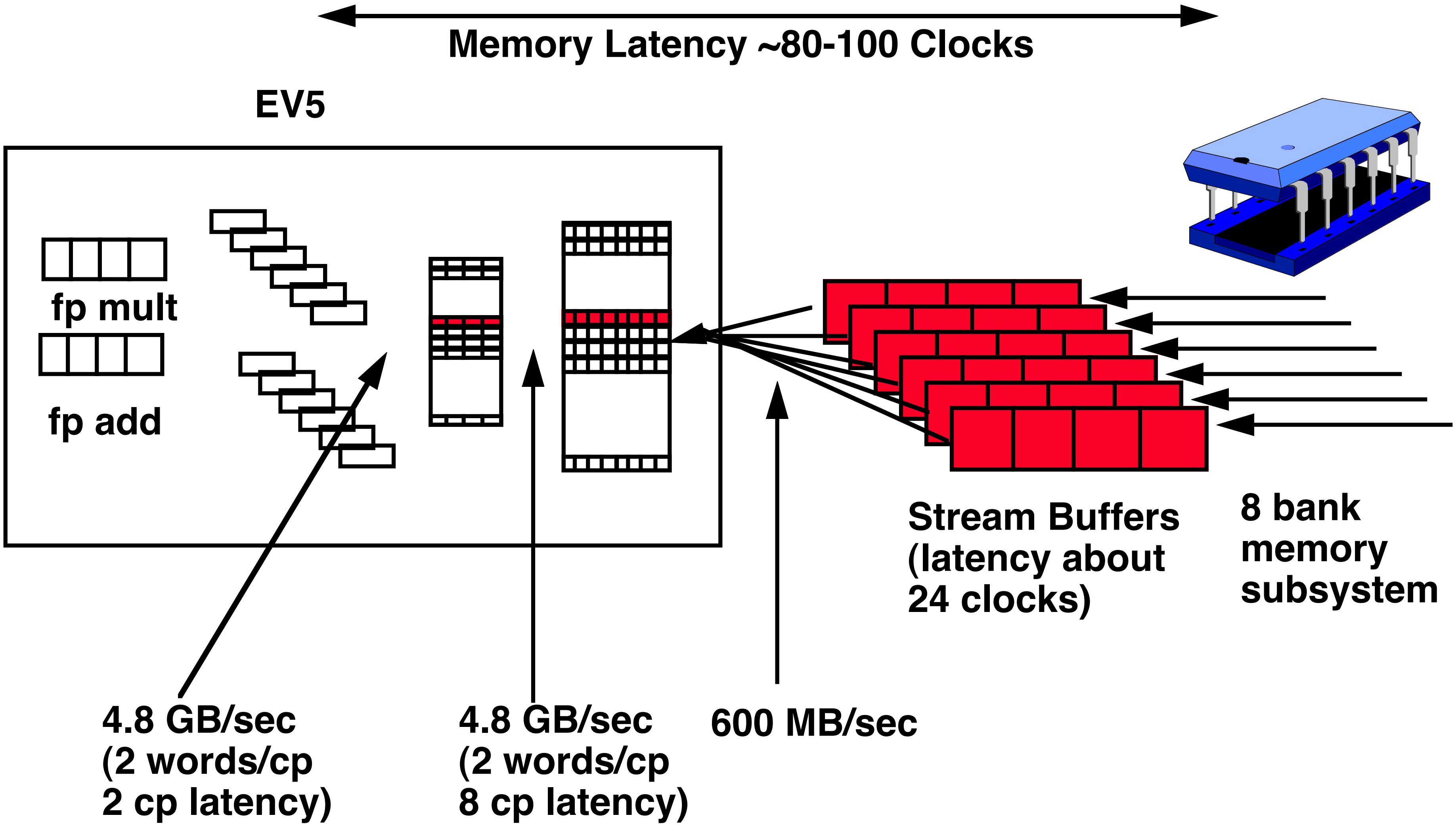


T3D System: 1 Read-Ahead Buffer
300 MB/sec Peak
120 MB/sec sustained on multiple RHS code

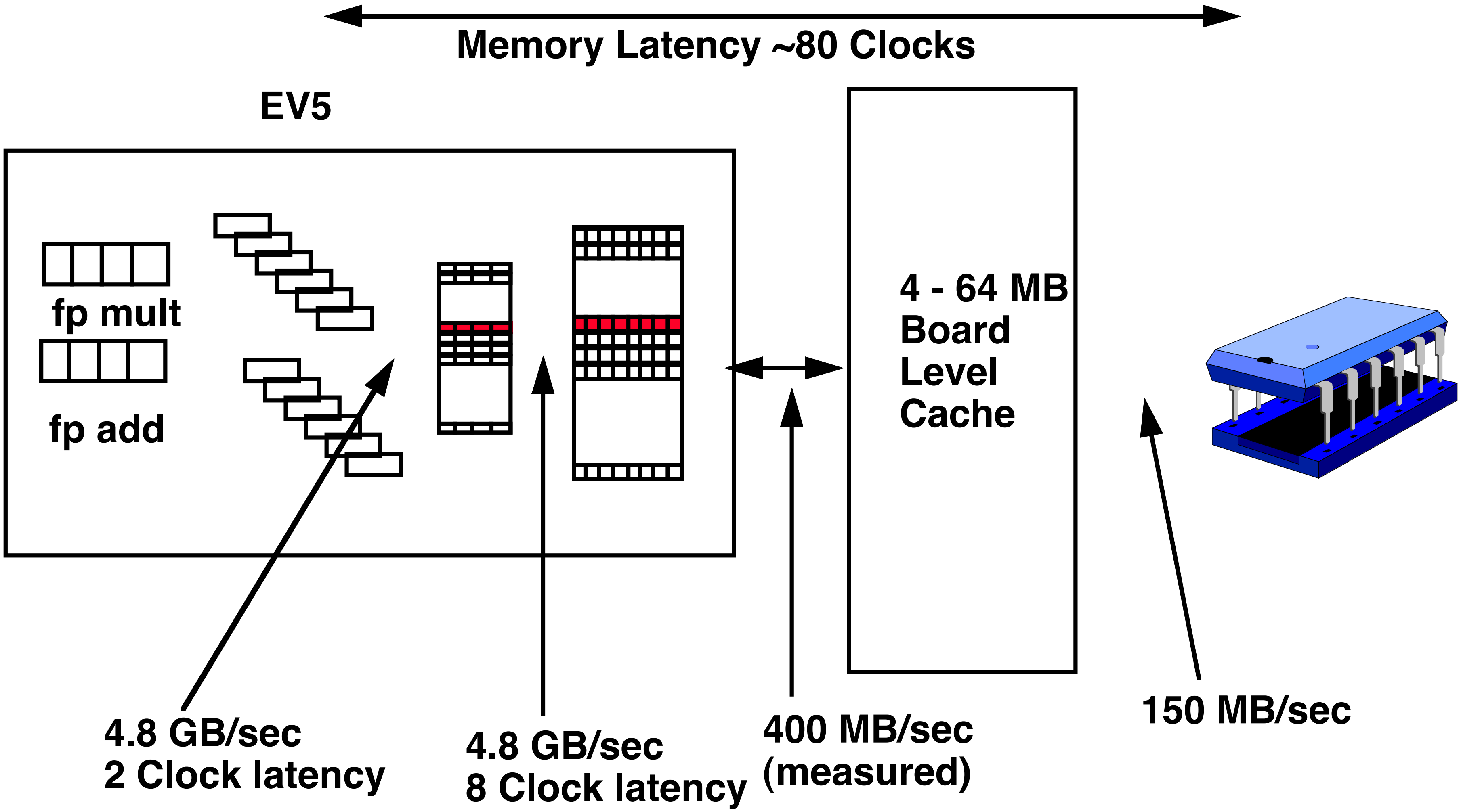


T3E System: 6 Stream Buffers
1 GB/sec Peak
600 MB/sec sustained on multiple RHS code

Cray T3E Node



DEC EV5 Workstation



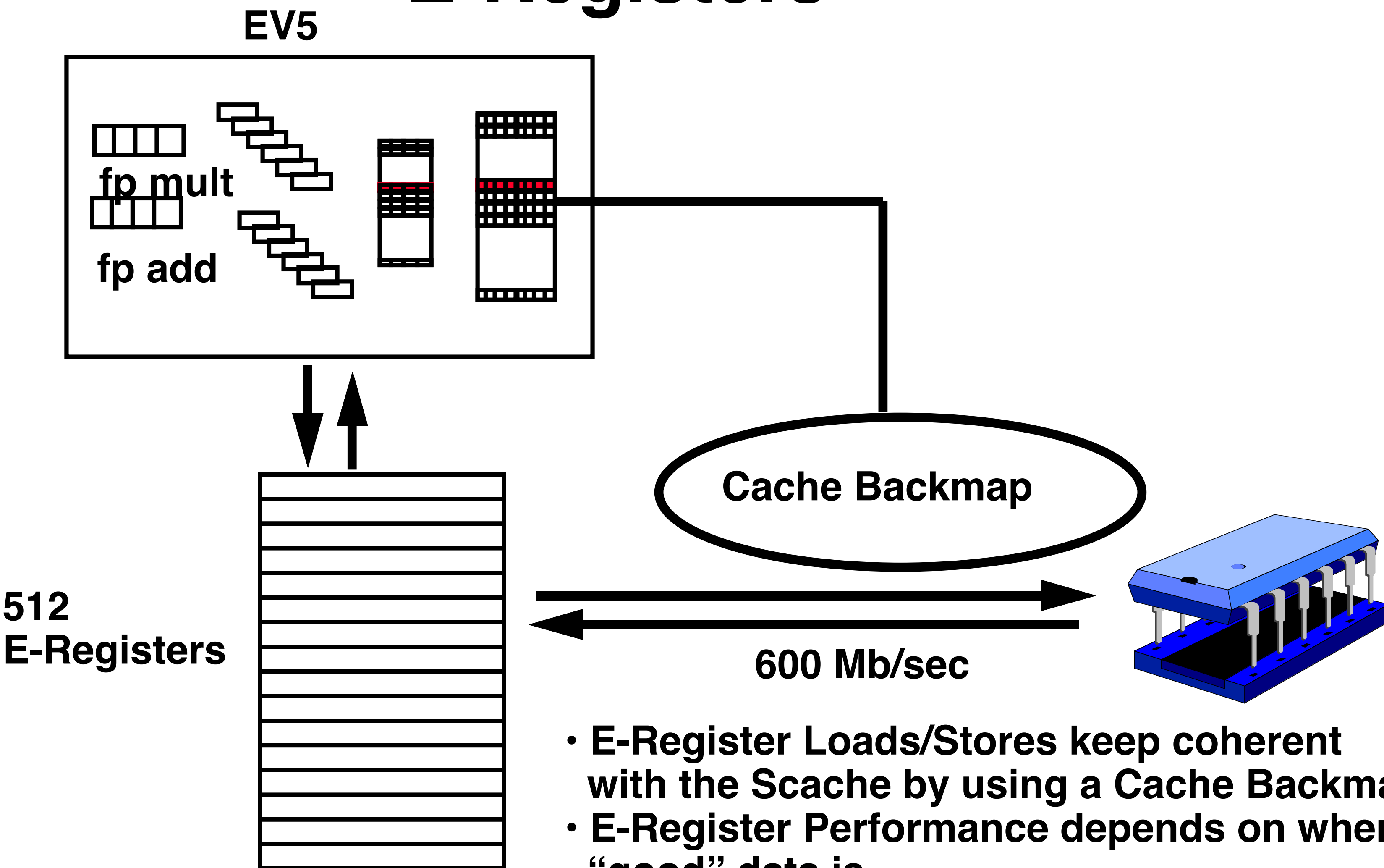
E-Registers

- E-registers are used for all remote memory access□
- E-registers integrate the CRAY T3D system discrete latency hiding and synchronization mechanisms into a single remote reference□ control mechanism□
  - Prefetch queue
  - Block transfer engine
  - Fetch and increment
  - Atomic swap
  - Hardware message□passing

Global Memory Access

- E-registers: 512 off-chip memory-mapped register
  - GET: load from global memory to E-register
  - PUT: store from E-register to global memory
- E-register data can be loaded into DEC Alpha 21164 registers at 600 MB/s

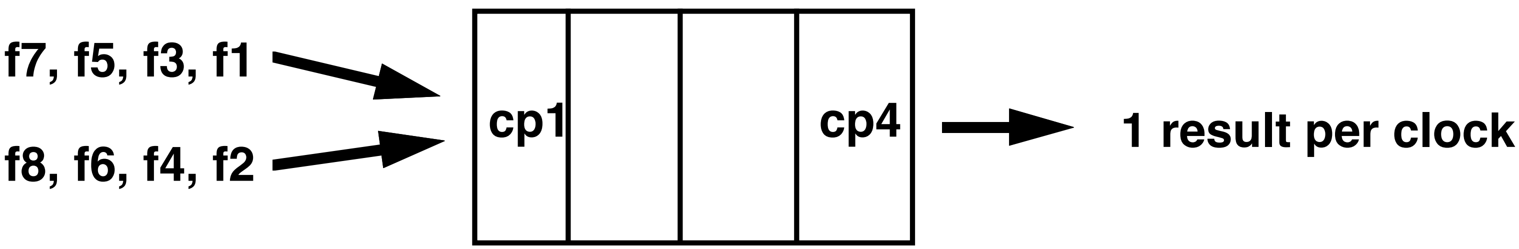
E-Registers



- E-Register Loads/Stores keep coherent with the Scache by using a Cache Backmap
- E-Register Performance depends on where “good” data is

FP Functional Units

Operation	T3D	T3E	Piplined?
FP Multiply	6 cp	4cp	Yes
FP Add	6 cp	4cp	Yes
FP Divide	61 cp	22-60 cp	No



Pipelined Add or Multiply

Functional Unit Latency

```
do i = 1, 1024
   t = t * t
enddo
```

- Expect 1 result per 4 clock periods or 75 Mflops.
- Observe 74.4 Mflops

```
do i = 1, 1024
   t1 = t1 * t1
   t2 = t2 * t2
   t3 = t3 * t3
   t4 = t4 * t4
enddo
```

- Expect near 300 Mflops
- Observe 216 Mflops

In general, unrolling loops will expose more functional unit parallelism to the compiler.

Functional Unit Latency

4th Order Horner's Rule Polynomial

```
do i = 1, 1024
y(i)=(((px4*x(i)+px3)*x(i)+px2)*x(i)+px1)*x(i) +px0
enddo
```

Condition	Performance		
	T3D	T3E	Ratio
AS IS	19.5	70	3.6x
Unroll by 4	38.2	250	6.5x
Unroll by 8	65.1	403	6.2x
Add Read-Ahead	78.1	403	5.1x

- Implications for intrinsic functions
- 50th order Horner's rule runs at 562 Mflops on the T3E (order a 2048 PE system, run at 1.1 Tflops and declare victory)

Intrinsic Functions

(All timings in clock periods per result)

Routine	libm			libbench vector n=1000		
	T3D	T3E	ratio	T3D	T3E	ratio
SQRT	235	82	5.7	34	32	2.1
1.0 / SQRT	298	116	5.1	36	26	2.8
ALOG	246	168	2.9	41	33	2.5
EXP	198	88	4.5	43	34	2.5
SIN	194	145	2.6	53	46	2.3
COS	214	136	3.1	53	45	2.4
A ** B	1415	832	3.4	122	62	3.9

- A** B is done with a formal replacement A ** B = EXP(B* ALOG(A))
- Libm routines are good to 1/2 ULP, Benchlib routines are good to 2 ULPS

Intrinsic Functions
Livermore Loop 22

- Problems are non-overlappable divide, and intrinsic function call
- Original Code: 29 Mflops
- Modified Code: 52 Mflops

Original Code

```
fw = 1.0
do 22 k = 1, 101
  y(k) = u(k) / v(k)
  w(k) = x(k) / (exp(y(k)) - fw)
enddo
```

Using Benchlib

```
real tv(101)
fw = 1.0
do k = 1, 101
  y(k) = u(k) / v(k)
enddo
call exp_v(101,y,y)
do k = 1, 101
  w(k) = x(k) / (y(k)-fw)
enddo
```

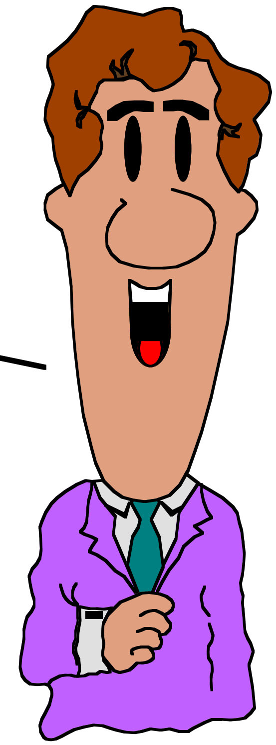
- Note that T3D timings were 7 and 23 Mflops respectively with two calls to “oneover_v” to do the divides

Intrinsic Functions

- Bnchlib still pays off on the T3E, but not as much due to shorter functional unit times.
- Bnchlib has not been rescheduled for the T3E quad issue yet.
- Bnchlib is accurate to 2 ulps vs. .5 ulps for system routines.
- f90 feature which automatically stripmines vector loops and calls vector intrinsics is on the way.
- The oneover_v() function is still faster, but just barely.

And now, a word from our Lawyers

This software is provided as is and Cray Research, Inc. disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness for a particular purpose. In no event shall Cray Research, Inc. be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this sofware



Saxpy

Code	Compilation
common /foo/ x(1024), y(1024) do i = 1, 1024 y(i) = y(i) + alpha * x(i) enddo	No Unrolling: f90 file.f Unrolling: f90 -Ounroll2 file.f

	T3D Mflops	T3E Mflops	T3E/T3D
No Unrolling	6.7	32.4	4.8 X
Unrolling	7.6	48.6	6.4 X

Saxpy

Code	Compilation	
common /foo/ x(1024), pad(512), y(1024)		
do i = 1, 1024	No Unrolling:	f90 file.f
y(i) = y(i) + alpha * x(i)	Unrolling:	f90 -Ounroll2 file.f
enddo		

	T3D Mflops	T3E Mflops	T3E/T3D
No Unrolling	10.8	32.6	3.0 X
Unrolling	14.5	62.3	4.3 X

- Direct-mapped primary cache padding has less effect on T3E
- Turning Streams off cuts performance by factor of 2X on the T3E

Streams Example

```
      real(kind=8) s8,a(102656,8)
      ...
      s8=0.
      do i=1,n
      s8=s8+a(i,1)+a(i,2)+a(i,3)+a(i,4)+
     $         a(i,5)+a(i,6)+a(i,7)+a(i,8)
      enddo
```

- The T3E supports up to 6 prefetching "streams" to memory
- 8 input streams require some modification...

Original Performance:	32.7 Mflops
Read Bandwidth:	262 Mbytes/sec

Streams Example (cont.)

```
      s8=0.
      do i1=1,n,256
      i2=min(i1+255,n)
      pntrs(1,1)=loc(a(i1,1))
      pntrs(2,1)=loc(a(i2,1))
      pntrs(1,2)=loc(a(i1,2))
      pntrs(2,2)=loc(a(i2,2))
      call PREFUM(2,pntrs)
      do i=i1,i2
      s8=s8+a(i,1)+a(i,2)+a(i,3)+a(i,4)
     $        +a(i,5)+a(i,6)+a(i,7)+a(i,8)
      enddo
      enddo
```

- Call to PREFUM prefetches two of the data streams
- Remaining 6 data streams are handled by the stream buffers

PreFetch Performance:	52.7 Mflops
Read Bandwidth:	422 Mbytes/sec

Streams Example (cont.)

```
s8=0.
do i=1,n
s8=s8+a(i,1)+a(i,2)+a(i,3)+a(i,4)
enddo
do i=1,n
s8=s8+a(i,5)+a(i,6)+a(i,7)+a(i,8)
enddo
```

PreFetch Performance:	74.1 Mflops
Read Bandwidth:	593 Mbytes/sec

- Other strategy is to break the loop up into two loops with 4 data streams each.
- This can be done by the compiler with -Osplit2
- More effective than prefetching strategy.

FDTD Example

```
      real ex(129,129,128), ey(129,129,128), ez(129,129,128)
      real hx(129,129,128), hy(129,129,128), hz(129,129,128)

      do j= 2, 127
         do i= 2, 127
            do k=2, 127
               ex(k,i,j) =
     .              caex*(ex(k,  i,j) +
     .              cbex*(hz(k,  i,j) - hz(k,i,j-1) +
     .                          hy(k,  i,j) - hy(k-1,i,j)))
               ey(k,i,j) =
     .              caey*(ey(k,  i,j) +
     .              cbey*(hx(k-1,i,j) - hx(k,  i,j) +
     .                          hz(k,i-1,j) - hz(k,  i,j)))
               ez(k,i,j) =
     .              caez*(ez(k,  i,j) +
     .              cbez*(hx(k,i,j-1) - hx(k,  i,j) +
     .                          hy(k,  i,j) - hy(k,i-1,j)))
            enddo
         enddo
      enddo
      return
      end
```

8 input streams
3 output streams

- Loop can be split manually, or with the -Osplit2 option on f90.

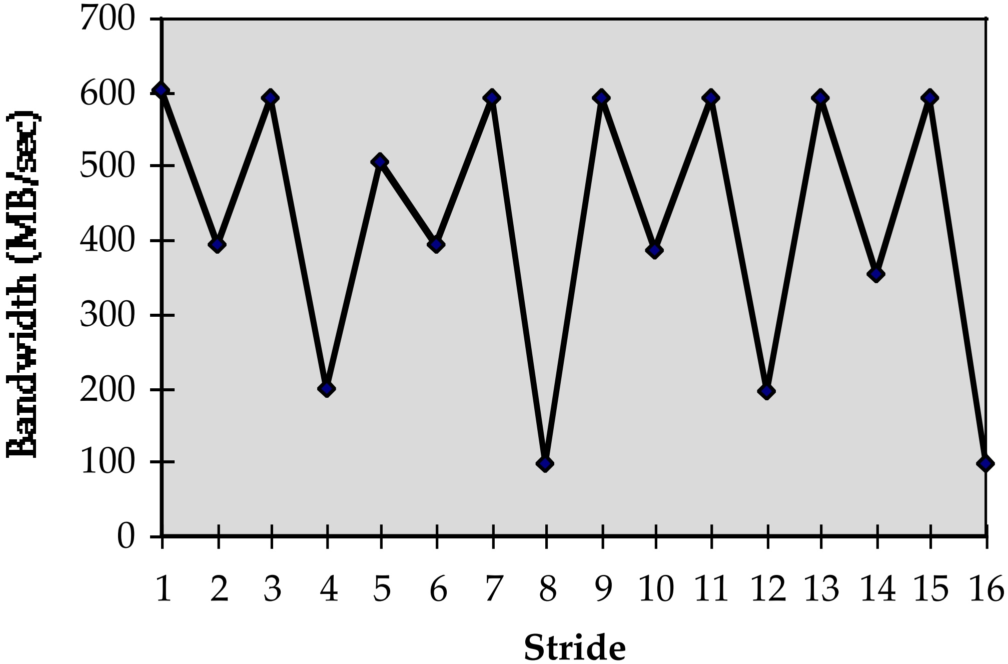
Performance (Mflops)		
T3D	T3E	T3Esplit
16.1	45.2	73.6

Bnchlib E-Register Routines

- LGETV(source, stride, length)
  - Gets data from memory to E-registers
- LPUTV(source, stride, lenght)
  - Puts data from E-registers to memory
- LSETV(target, stride, length, value)
  - Sets array to value without modifying Scache
- LPUTP()
  - Waits for E-register puts to complete
- LGATH(array, index, lenght)
  - Gathers from memory into E-registers
- LSCAT(array, index, lenght)
  - Scatters data from E-registers to memory

- Note: In all cases, length must be .LE. 480 due to number of E-registers

E-Register Bandwidth with LGETV



- Note that this is bandwidth from memory into E-Registers
- An additional step is required to read data into EV5
- Clearly shows 8-bank memory
- Very large odd-strides decay a bit due to page misses.
- If all data is in Scache, results decay by < 2X

E-Register Transpose

- It is possible to use E-registers to do a fast local transpose.

```
real x(401,400), xt(401,400)
do i = 1, 400
  do j = 1, 400
   xt(i,j) = x(j,i)
  enddo
enddo
```

Cached fortran runs at 126 MB/sec

```
real x(401,400), xt(401,400)
c  transpose w strided GETV
do j=1,401
 call lgetv(x(j,1),401,400)
 call lputv(xt(1,j),1,400)
enddo
```

E-register version runs at 438 MB/sec

- Arbitrarily large transposes can be written with strip-mining
- Note that data never was in EV5 chip!

E-Register Strided SDOT

```
real a(1025,1024), b(1024)
do i = 1, 1024
  x(i) = sdot(1024, a(i,1), 1025, b, 1)
enddo
```

Fortran Strided SDOT: 17.5 Mflops

```
      real a(1025,1024), b(1024)
      pointer (l_ereg,eregs(0:511))
      integer*8 eregs,t
c The ereg array pointer is initialized as follows:
      l_ereg=shiftl(15,32)
      do i = 1, 1024
        s = 0.0
        do j = 1, 1024,400
         istrip = min(400,1024-j+1)
         call lgetv(a(i,j),1025,istrip)
         s = s +  sdot(istrip,eregs(32),1,b,1)
        enddo
        x(i) = s
      enddo
```

E-Register Strided SDOT: 42 Mflops

Conjugate Gradient Example

```
    do 200 i = 1, nrows
     y(i) = 0.
     do k = rowstr(i), rowstr(i+1) - 1
        y(i) = y(i) + a(k) * x (colidx(k))
     end do
200   continue
```

Sparse subset of X is gathered for each row of “a”

- Performance highly dependent on size of X array and resulting cache hit rate.
- For the CG benchmark, the T3D achieved about 5-6 Mflops on this code segment for both Class A and B sizes (cache was too small to help)

Conjugate Gradient via E-registers

```
      pointer (l_ereg,eregs(0:511))
      real*8 eregs
      l_ereg=shiftl(15,32)

      do 200 i = 1, nrows
        ytmp=0.
        do k1=rowstr(i),rowstr(i+1)-1, 480
            k2= min(k1+479, rowstr(i+1)-1)
            call lgathh(x(0),colidx(k1),k2-k1+1)
            ytmp=ytmp+sdot(k2+1-k1,eregs(32),1,a(k1),1)
        enddo
        y(i)=ytmp
 200   continue
```

- Call to lgathh gathers 480 values of X into E-registers which can be passed to SDOT.
- Performance should be independent of size of X.
- LGATHH runs at ~450 MB/sec

Conjugate Gradient Results

Class A Problem on 4 PEs:

Packed Storage A

7000x7000

X vector

7000

Standard Matvec: 24.7 Mflops/PE
E-Register Matvec: 27.0 Mflops/PE

Class B Problem on 16 PEs:

Packed Storage A

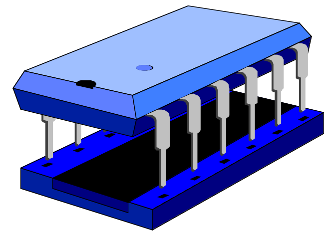
18750x18750

X vector

18750

Standard Matvec: 12.4 Mflops/PE
E-Register Matvec: 25.0 Mflops/PE

Summary



In General:

- **STRIDE 1 important**
- **Keep number of streams to 6 or less.**
- **Optimize for secondary cache use.**
- **Remember that stores may count as two streams because of Scache policy.**
- **Try E-registers for strided address computations and for gather/scatter operations that are not likely to be cache resident.**
- **Unrolling will help with Miss Address File utilization and functional unit parallelism.**
- **Bnchlib instrinsics**