T3E Individual Node Optimization

Michael Stewart, SGI/Cray, 4/9/98

- Introduction
- T3E Processor
- T3E Local Memory
- Cache Structure
- Optimizing Codes for Cache Usage
- Loop Unrolling
- Other Useful Optimization Options
- References

Introduction

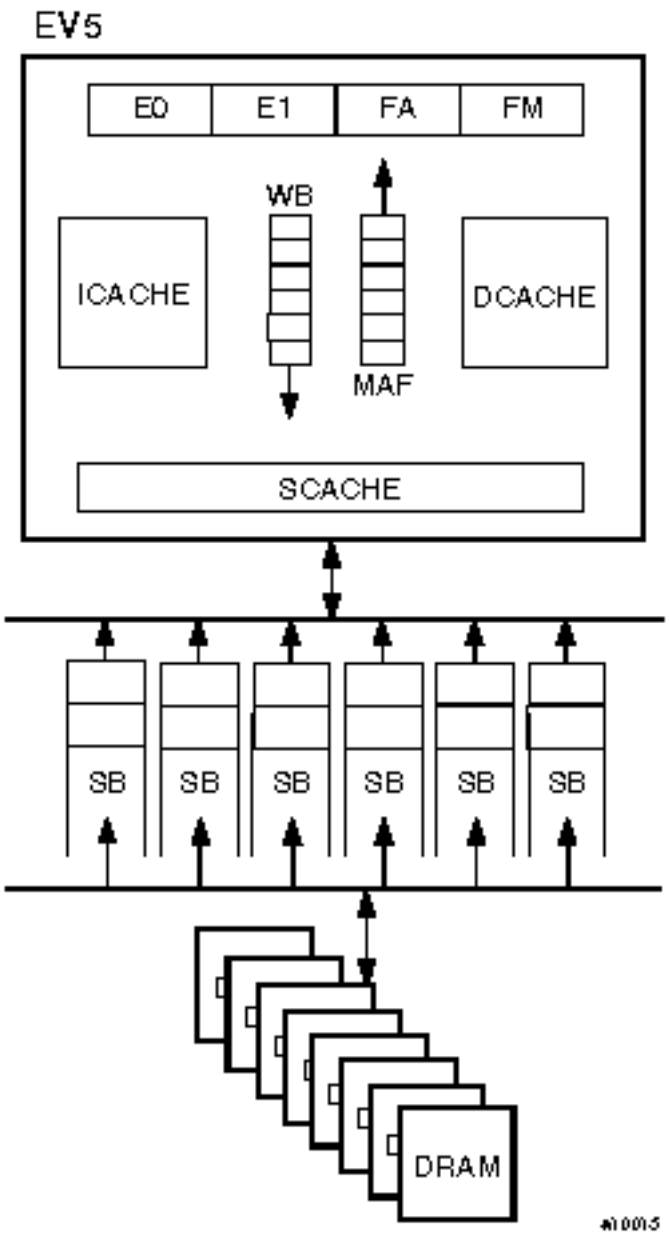
- Primary topic will be single processor optimization
  - Most codes on the T3E are dominated by computation
  - Processor interconnect specifically designed for high performance codes, unlike the T3E processor
- More detailed information available on the web (see References)
- Fortran oriented, but I will give C compiler flag equivalents.

T3E Processor

- Commodity DEC Alpha EV5 RISC microprocessor, 1 FP add and 1 FP multiply per CP (clock period)
  - mcurie: 450 MHz clock, 900 MFlops peak performance
  - pierre: 300 MHz clock, 600 MFlops peak performance
- All puts and gets between processors are cache coherent
- Principle differences between T3E and traditional Cray PVP's (Parallel vector processors):
  - T3E strictly scalar, no vector registers
  - T3E has a hierarchical memory structure with several layers of caching
  - T3E optimization more data dependent

T3E Local Memory

- Hierarchical memory with two "and a half" levels of data caching
- On chip primary and secondary cache plus Cray provided "data streaming"
- Memory access time ranges from 2-125 CP (clock periods)



Data Flow on a T3E Node

DRAM – Local Memory

SB – Stream Buffers

SCACHE – Secondary Cache

DCACHE – Primary Cache

ICACHE – Instruction Cache

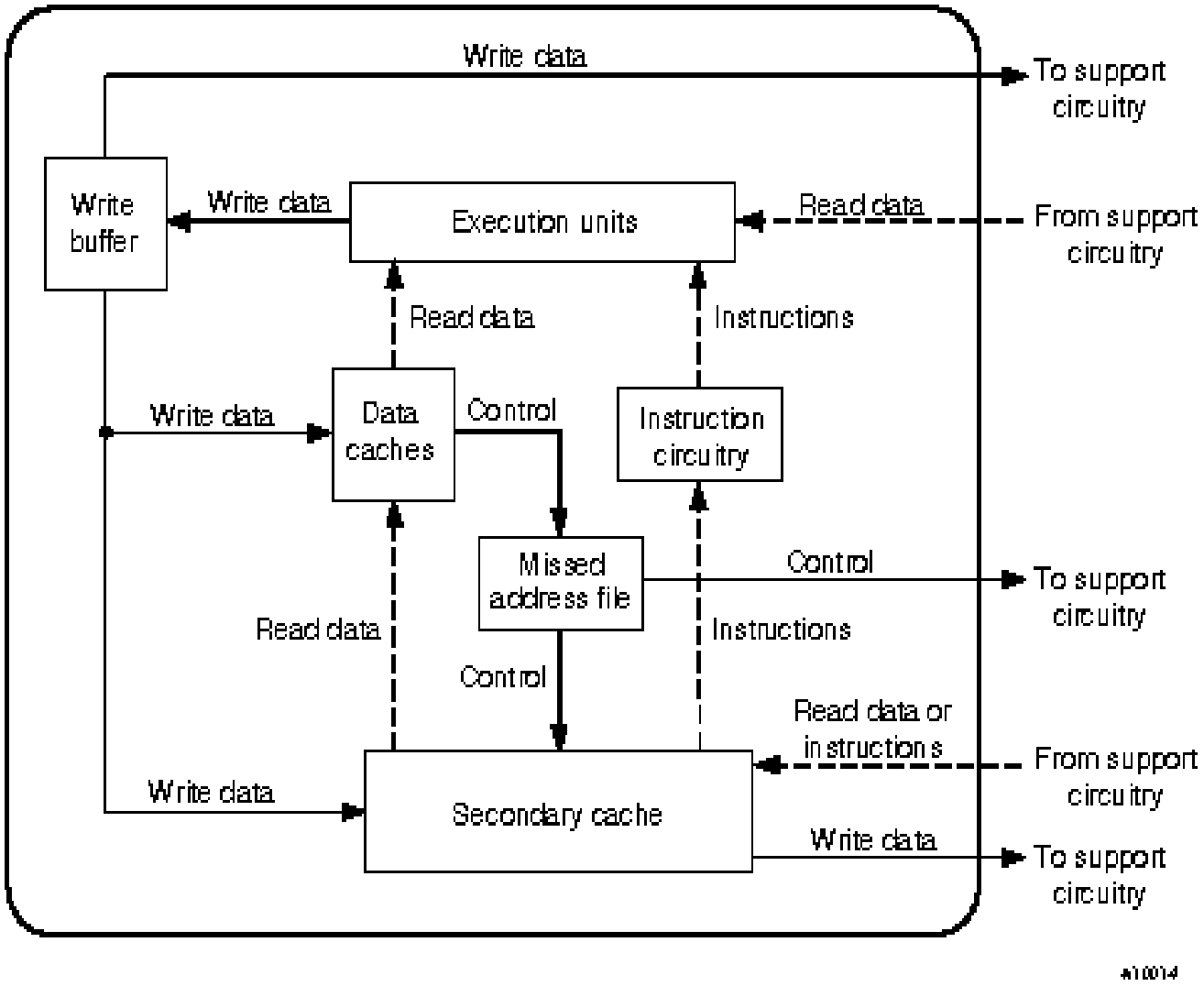
MAF – Missed Address File

WB – Write Buffer

E0/E1 – Integer Functional Units

FA/FM – FP Functional Units

Data Flow on the EV5 Processor



Primary Cache ("Data Cache")

- Direct Mapped
  - Each memory location is mapped to a specific cache line
  - Many to one mapping
- Basic unit is 4 64 bit word line
- Total size 8 KBytes (256 lines)
- Latency – 2 CP per load
- Bandwidth – 2 words per CP

Secondary Cache

- 3 way set associative
  - Each memory location is associated with 3 different cache lines
  - One of the 3 lines is chosen at random when data is loaded into cache
- Basic unit 8 64 bit word line
- Size 96 (3x32) KBytes, 3x512 lines
- Latency – 8 CP per load
- Band width – 2 words per CP

Stream Buffers

- Prefetch secondary cache lines into buffers
- Streaming begins when there are two consecutive secondary cache line misses
- Can have up to 6 streams active
- Latency – 24 CP per load
- Bandwidth – .36 word per CP

Local Memory

- 256 MBytes (32 MWords) per processor on mcurie
- 128 MBytes (16 MWords) per processor on pierre
- Latency – 85/125 CP per load (page hit/miss)
- Band Width – .08/.06 word per CP

Optimizing Code for Cache Usage (1)

- Increase the amount of time data are used while in cache by maximizing loop invariant references in innermost loop
  - C and D are reloaded with each iteration of the inner loop:

```
DO I=1,ILIM
  DO J=1,JLIM
    DO K=1,KLIM
      A(I,J,K)=B(I,J,K)+C(J,K)+D(K,J)
    ENDDO
  ENDDO
ENDDo
```

  - C and D are now loop invariant in the innermost loop:

```
DO K=1,KLIM
  DO J=1,JLIM
    DO I=1,ILIM
      A(I,J,K)=B(I,J,K)+C(J,K)+D(K,J)
    ENDDO
  ENDDO
ENDDO
```

Optimizing Code for Cache Usage (2)

- Reduce cache conflict by "padding arrays"

  Cache conflict between B and C:

```
      COMMON /AAA/ A(1024), B(1024), C(1024)

      DO I=1,1024
        A(I) = B(I) + C(I)
      ENDDO
```

  No cache conflict:

```
      COMMON /AAA/ A(1024), B(1024), PAD(4), C(1024)
!DIR$ CACHE_ALIGN /AAA/
```

- -a pad[n] flag to f90 automatically pads arrays but violates

  Fortran addressing conventions

Loop Unrolling

- Increase the amount of work done in a single iteration of a do loop to reduce loop overhead and make best use of the T3E's segmented functional units by "pipelining" operands.

```
    DO I = 1, iter
      T = T + T
    ENDDO

  Unrolled:

    DO I = 1, iter/4
      T1 = T1 + T1
      T2 = T2 + T2
      T3 = T3 + T3
      T4 = T4 + T4
    ENDDO

    T = T1+T2+T3+T4
```