[Microprocessor Applications]

Lab 5: Cache Optimization

Chester Sungchung Park

SoC Design Lab, Konkuk University

Webpage: http://soclab.konkuk.ac.kr

Outline

- Objectives
- Description
- Block diagram
- Source codes
- Evaluation

Objectives

- Optimizing a C application whose performance is dominated by memory access by reducing the number of cache misses

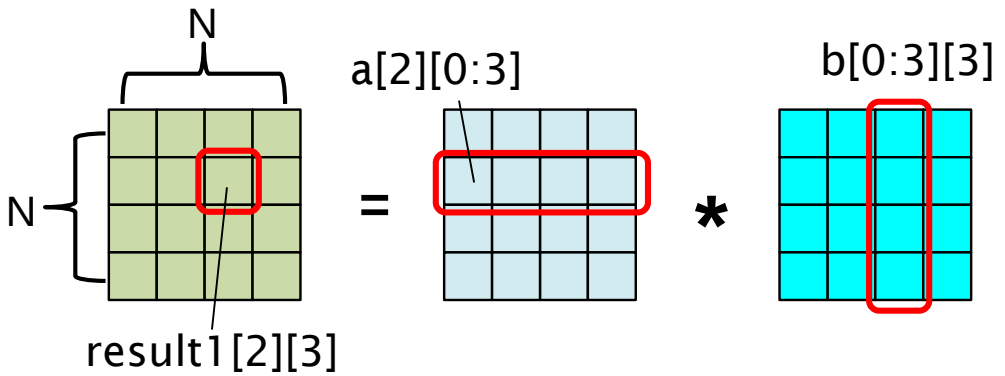
Description

- Matrix multiplication **without** tiling

```
for(i=0;i<N;i++)
   for(j=0;j<N;j++)
      for(k=0;k<N;k++)
         result1[i][j] += a[i][k]*b[k][j];
```

Ex. N=4, T=2

i=2, j=3, k=0:3

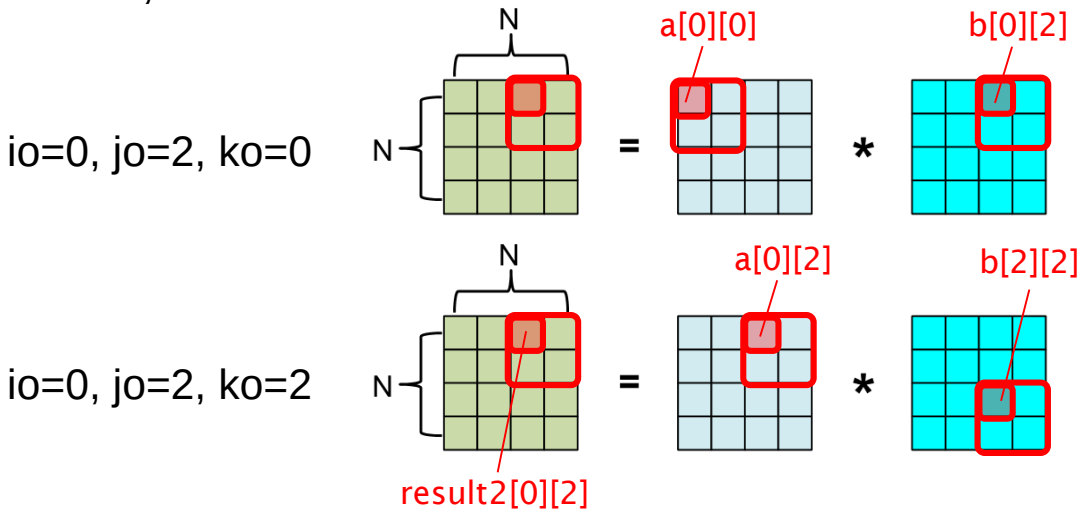


Description

- Matrix multiplication **with** tiling

```
for (io = 0; io < N; io += T)
    for (jo = 0; jo < N; jo += T)
        for (ko = 0; ko < N; ko += T)
            for (ii = 0, rresult = &result2[io][jo], ra = &a[io][ko];
                 ii < T; ii++, rresult += N, ra += N)
                for (ki = 0, rb = &b[ko][jo];ki < T; ki++, rb += N)
                    for (ji = 0; ji < T; ji++)
                        rresult[ji] += ra[ki] * rb[ji];
```

Ex. N=4, T=2



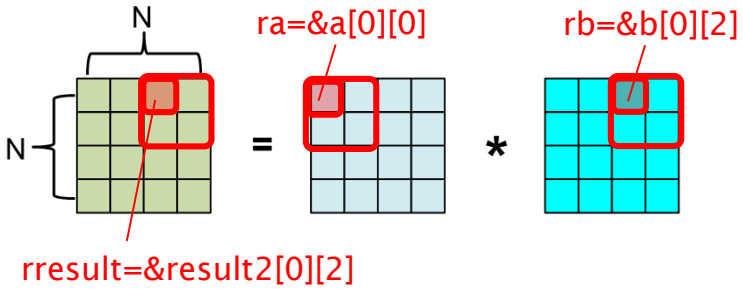
Description

- Matrix multiplication **with** tiling (cont'd)

```
for (io = 0; io < N; io += T)
    for (jo = 0; jo < N; jo += T)
        for (ko = 0; ko < N; ko += T)
            for (ii = 0, rresult = &result2[io][jo], ra = &a[io][ko];
                 ii < T; ii++, rresult += N, ra += N)
                for (ki = 0, rb = &b[ko][jo];ki < T; ki++, rb += N)
                    for (ji = 0; ji < T; ji++)
                        rresult[ji] += ra[ki] * rb[ji];
```

Ex. N=4, T=2 (io=0, jo=2, ko=0)

io=0, jo=2, ko=0

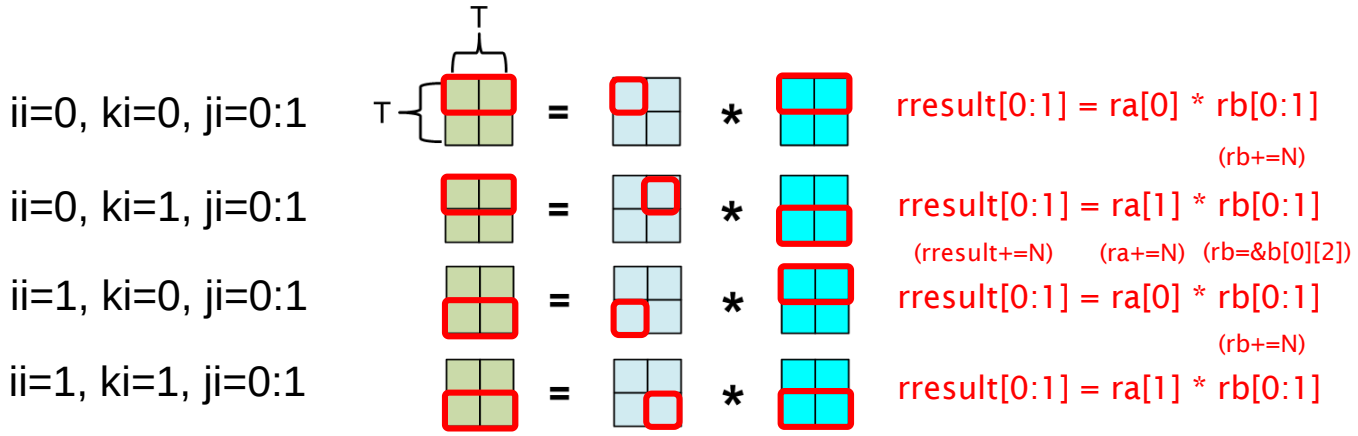


Description

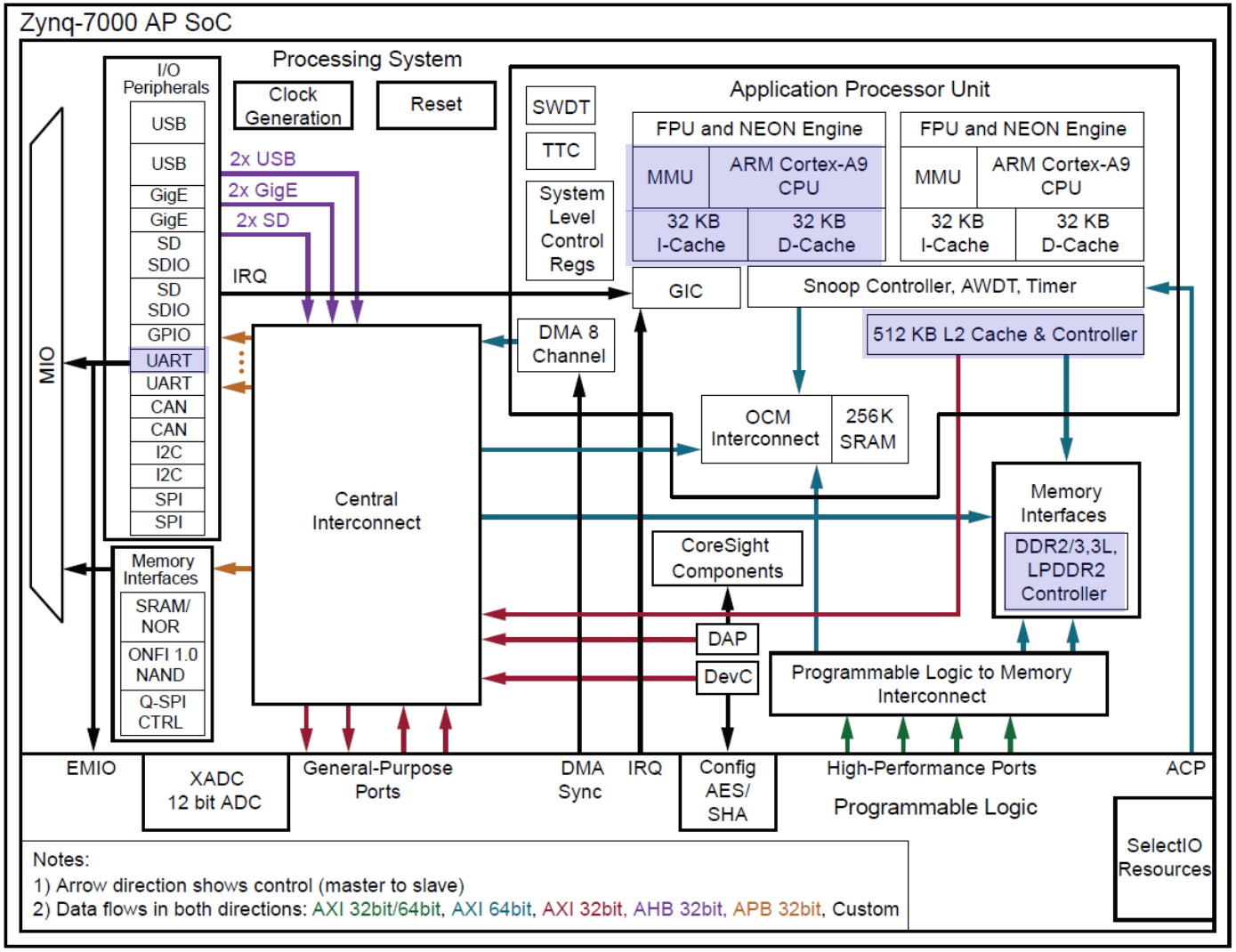
- Matrix multiplication **with** tiling (cont'd)

```
for (io = 0; io < N; io += T)
    for (jo = 0; jo < N; jo += T)
        for (ko = 0; ko < N; ko += T)
            for (ii = 0, rresult = &result2[io][jo], ra = &a[io][ko];
                 ii < T; ii++, rresult += N, ra += N)
                for (ki = 0, rb = &b[ko][jo];ki < T; ki++, rb += N)
                    for (ji = 0; ji < T; ji++)
                        rresult[ji] += ra[ki] * rb[ji];
```

Ex. N=4, T=2 (io=0, jo=2, ko=0)

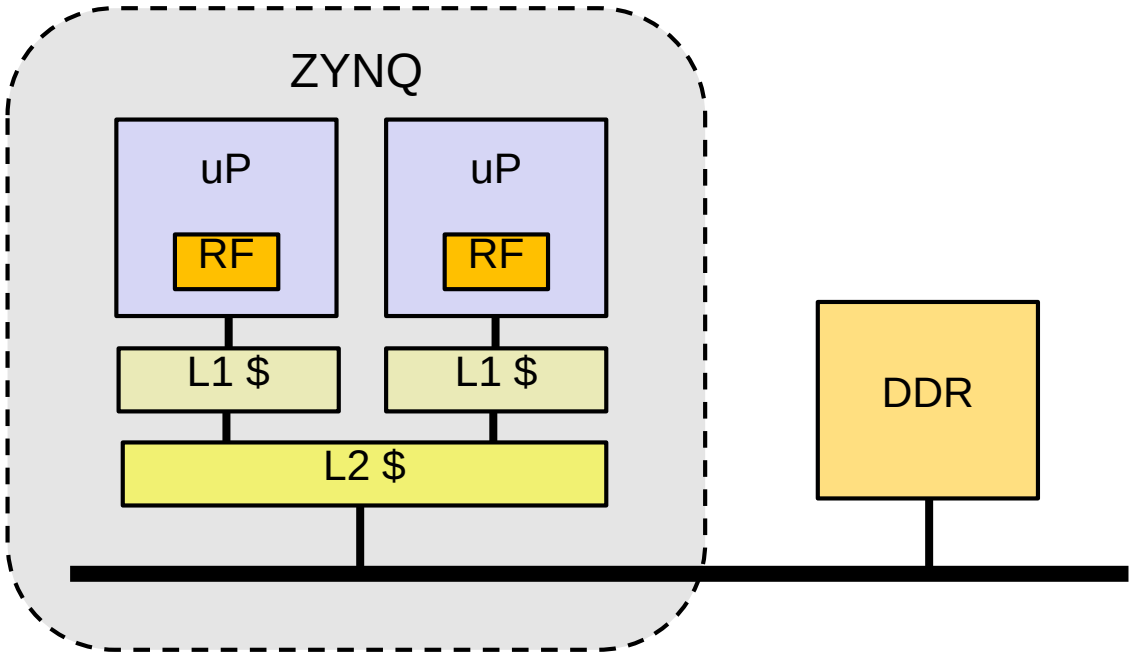


Block Diagram



Block Diagram

- Latency comparison (pipeline: 1x)
  - RF/L1 cache: 1x
  - L2 cache: 25x
  - DDR: 65x

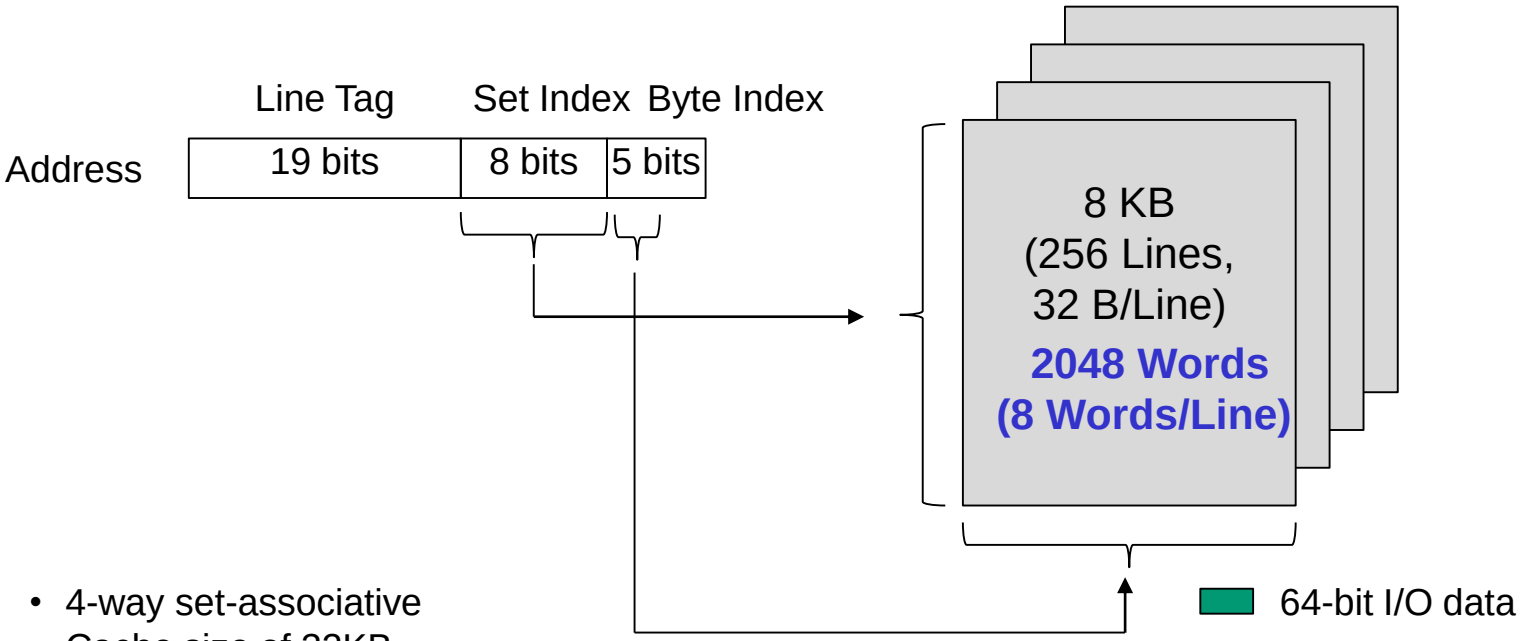


Block Diagram

- L1 cache
  - One for instruction and another for data per processor
  - Separate instruction and data caches
  - 4-way set-associative
  - Cache size of 32KB
  - Cache line size of 32 bytes (8 words)
  - Cache replacement policy of pseudo round-robin or pseudo random
  - Only write-back/write-allocation supported for data
    - ✓ Neither write-through nor write-back/read-allocation
  - Exclusive w.r.t. L2 cache for data
    - ✓ At any time, a given address is cached in either L1 data cache or L2 cache, but not in both

Block Diagram

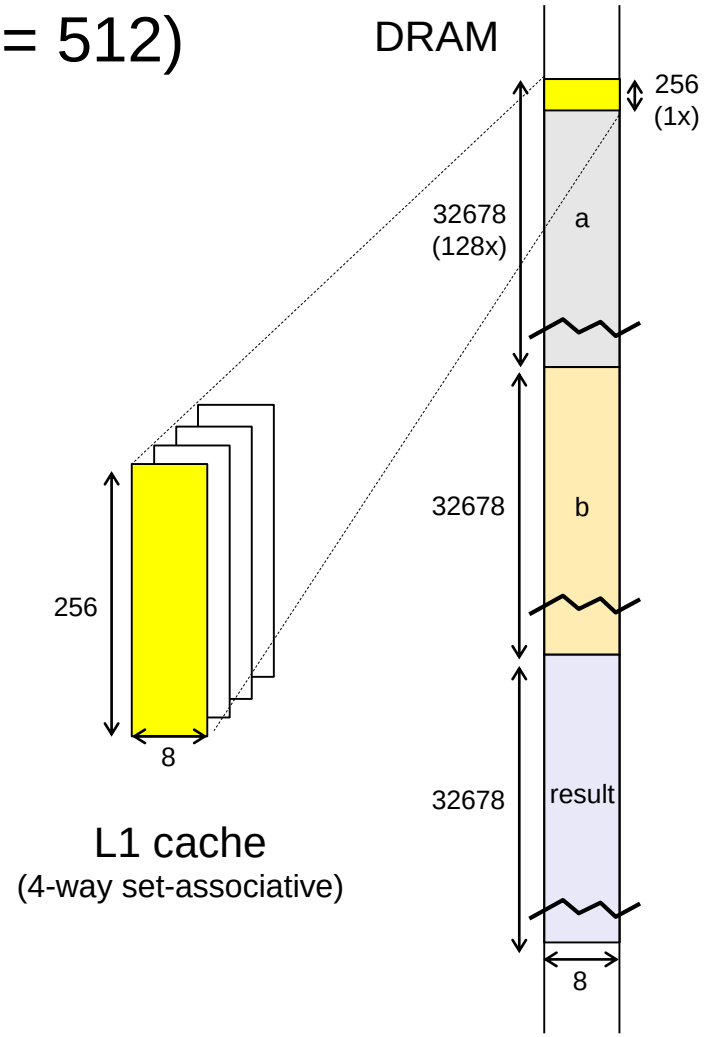
- L1 cache (cont'd)



- 4-way set-associative
- Cache size of 32KB
- Cache line size of 32 bytes (8 words)
- 64-bit data paths throughout memory system

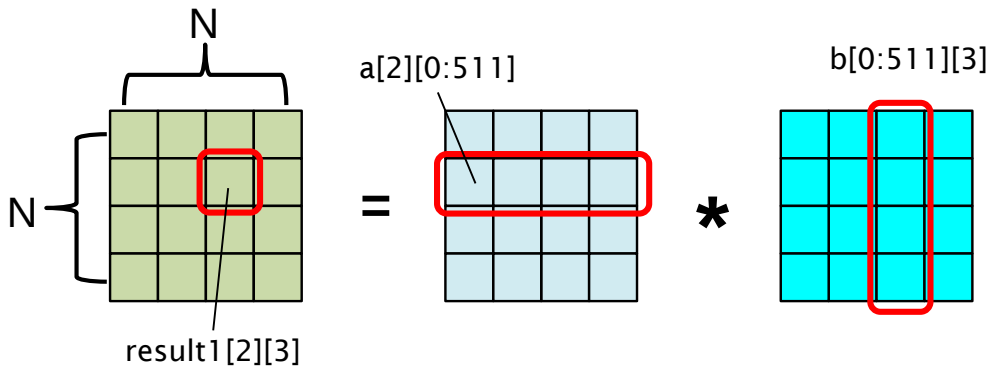
DRAM vs. Cache

- Example (N = 512)

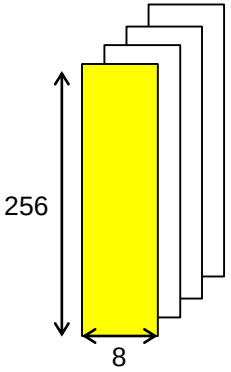


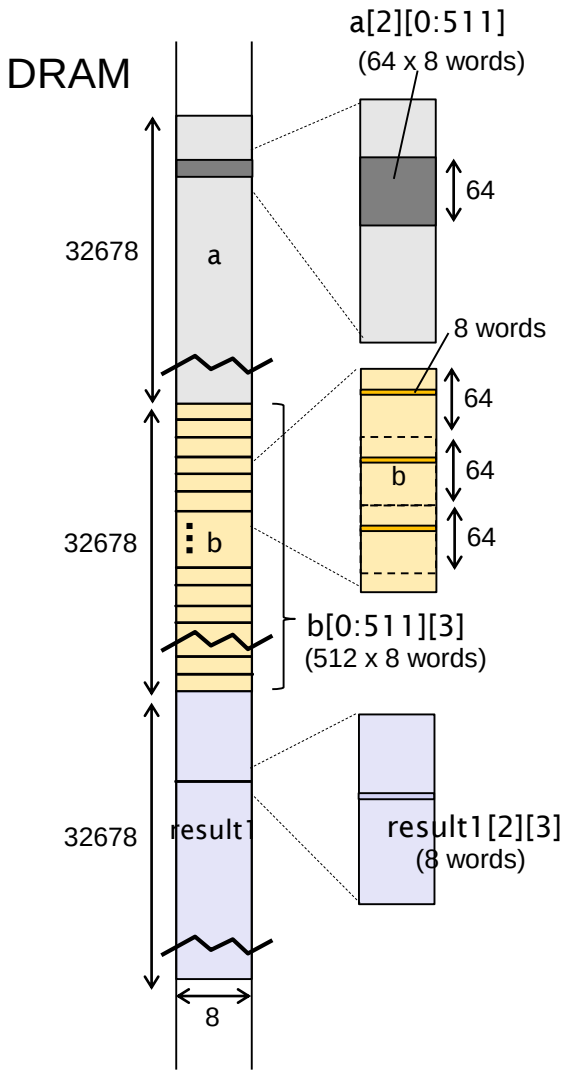
DRAM Data Layout

- Matrix multiplication **without** tiling
  - N multiplications (N = 512)



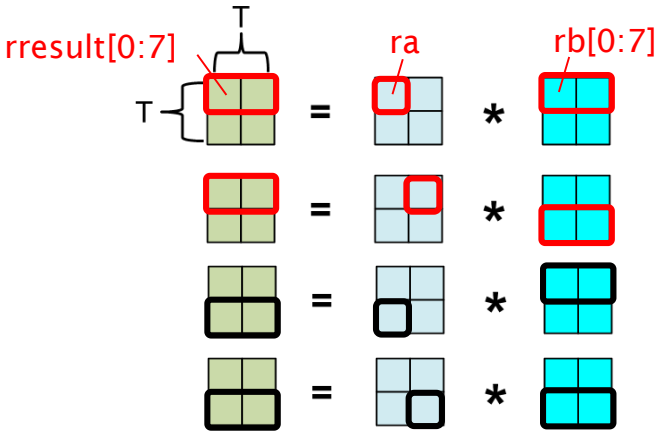
L1 cache
(4-way set-associative)



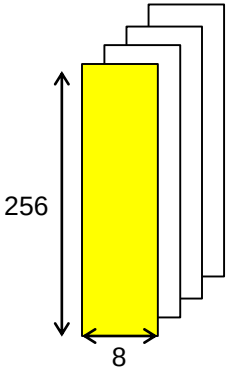


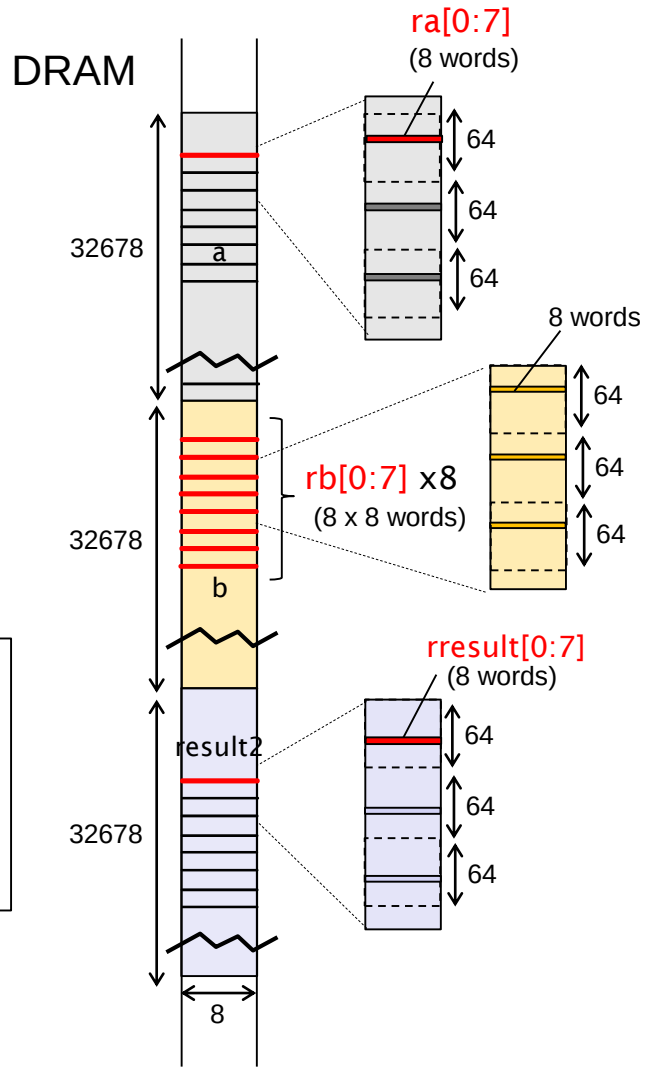
DRAM Data Layout

- Matrix multiplication **with** tiling
  - T^3 multiplications (N = 512, T = 8)



L1 cache
(4-way set-associative)





Source Codes

- main.c

benchmarking.c | benchmarking.h | main.c

```
int a[N][N],b[N][N],result1[N][N],result2[N][N];

unsigned mat_mult(unsigned int uiParam0, unsigned int uiParam1, unsigned int uiParam2, unsigned int uiParam3 )
{
    int i,j,k;

    for(i=0;i<N;i++)
       for(j=0;j<N;j++)
          for(k=0;k<N;k++)
             result1[i][j] += a[i][k]*b[k][j];

    return 1;
}

unsigned mat_mult_tiling(unsigned int uiParam0, unsigned int uiParam1, unsigned int uiParam2, unsigned int uiParam3 )
{
    int io,jo,ko,ii,ki,ji;
    int *rresult, *rb, *ra;

    for (io = 0; io < N; io += T)
        for (jo = 0; jo < N; jo += T)
            for (ko = 0; ko < N; ko += T)
                for (ii = 0, rresult = &result2[io][jo], ra = &a[io][ko];
                     ii < T; ii++, rresult += N, ra += N)
                    for (ki = 0, rb = &b[ko][jo];ki < T; ki++, rb += N)
                        for (ji = 0; ji < T; ji++)
                            rresult[ji] += ra[ki] * rb[ji];

    return 1;
}
```

References

- Zynq-7000 All Programmable, technical reference manual, Xilinx UG585