1. (Bonus 1) System performance (8p)

A system has the following properties

- 10% of the instructions are stores
- 20% of the instructions are loads

- 90% of the loads hit in the 1st level data cache
- 90% of the load accesses to the 2nd level cache hit there

- 90% of the stores hit in the 1st level data cache
- 70% of the store accesses to the 2nd level cache hit there
- All the instruction fetches hit in the separate L1 instruction cache
- The CPU is a "perfect" 1.0 CPI pipeline, if all the loads and stores hit in the 1st level cache.

- The CPU clock frequency is 100MHz, in-order, and has an infinitely large write buffer.
- There is a 10 cycle stall if you miss in L1 data cache and hit in the L2 cache.
- There is a 100 cycle stall if you miss in both L1 and L2 caches
- There is unlimited bandwidth at all levels of the hierarchy
- The cache line size is 64 bytes

a) What is the CPI for the system if the overhead in the memory system is also taken into account? (3p)
Only the loads will effect the CPI, since the stores are handled by the store buffer.
The load overhead is 0.2 * ((0.9 * + 0.1* (0.9*10+ 0.1*100)) = 2.8 * 0.2 = 0. 56
CPI = 0.7 (alu) + 0.1 (stores) + 0.56 (load OH)= 1.36

b) What is the bandwidth of all the memory traffic assuming a copy-back cache strategy and that 20% of the L2 misses causes a write-back (and that there is inclusion between the L1 and L2)? (3p)
With a CPI of 1, 100M instruction are executed each second. Here, CPI=1.36 => 100M/1.36 = 73Minstruct/s
Probability that an instr. cause a store miss: 0,1 * 0.1 * 0.3 = 0.003
Probability that an instr. cause a load miss: 0.2 * 0.1 * 0.1 = 0.002
Probability that an instr. cause a writebacks: (P(loads)+P(stores))*0.2 = 0.001
Probability that an instr. cause bus trans = P(load)+P(store)+P(WB)= 0.006
Each transaction is 64B
Bandwidth = 0.006 * 64B * 73M istr = 28 MB/s

c) What is the speedup if the second-level cache was made larger and thus its miss rate was cut in half?(1p)
The component associated with stores and ALU take half as long
Old exec time ~ 0.7 (alu) + 0.1 (stores) + 0.56 (load OH)= 1.36
The new CPI overhead is thus 0.2 * ((0.9 * + 0.1* (0.95*10+ 0.05*100)) = 2.35 * 0.2 = 0. 47 => new exec time=0.8+0.47 = 1.27
Speedup=1.36/1.27 = 1.07

d) What would be the speedup if instead the CPU was clocked twice as fast, but the memory and cache latency (measured in time) stayed the same? (1p)
Old exec time ~ 0.7 (alu) + 0.1 (stores) + 0.56 (load OH)= 1.36
New exec time ~0.35 + 0.05 + 0.56 = 0.96
Thus, the speedup is 1.36/0.96 = 1.41

2. (Bonus 2) Loop scheduling (8p)

Consider the loop and the corresponding compiler-generated code

```
for (i=1; i<=1000; i = i+1)
     x[i] = x[i] + x[i+1];
```

```
loop:   LDD F0, 0(R1)       ;line 1
        LDD F2, -8(R1)      ;line 2
        ADDD F4, F0, F2     line 3
        SDD 0(R1), F2       ;line 4
        SUBI R1, R1, #8     ;line 5
        BNEZ R1, loop       ;line 6
```

*) LDD = Load double, SDD=store double

Assume that the array is stored in "backward order" in the array, i.e., that the address on x[i+1] is 8 bytes smaller than the address of x[i].

The pipeline has the following characteristics	Delay
INT ALU OP OUTPUT to INT ALU OP INPUT:	0 cycles
FP ALU OP OUTPUT to FP ALU INPUT:	2 cycles
LD DATA to INT/FP INPUT (load delay)	2 cycles
FP ALU OUTPUT to ST INPUT:	2 cycles
Branch delay slots:	2 cycles

a) Write one-sentence comments for each line (2p)

Line1: F0, F1 is now x[i]
Line2: F2,F3 is now x[i+1]
Line3: Store the sum in F4,5
Line4: Write the result to memory x[i]
Line5: Decrement array pointer
Line6: If array pointer != 0, loop back to line 1

b) Show where the bubbles are in each and calculate the number of cycles in each iteration? (2p)

```
loop:   LD F0, 0(R1)          ;line 1
        LD F2, -8(R1)         ;line 2
        stall
        stall
        ADDD F4, F0, F2       ;line 3
        stall
        stall
        SD 0(R1), F4          ;line 4
        SUBI R1, R1, #8       ;line 5
        BNEZ R1, loop         ;line 6
        stall (or NOP)
        stall (or NOP)
```

c) Show how the loop can be statically scheduled to improve the performance and calculate the number of cycles required for each iteration. (2p)

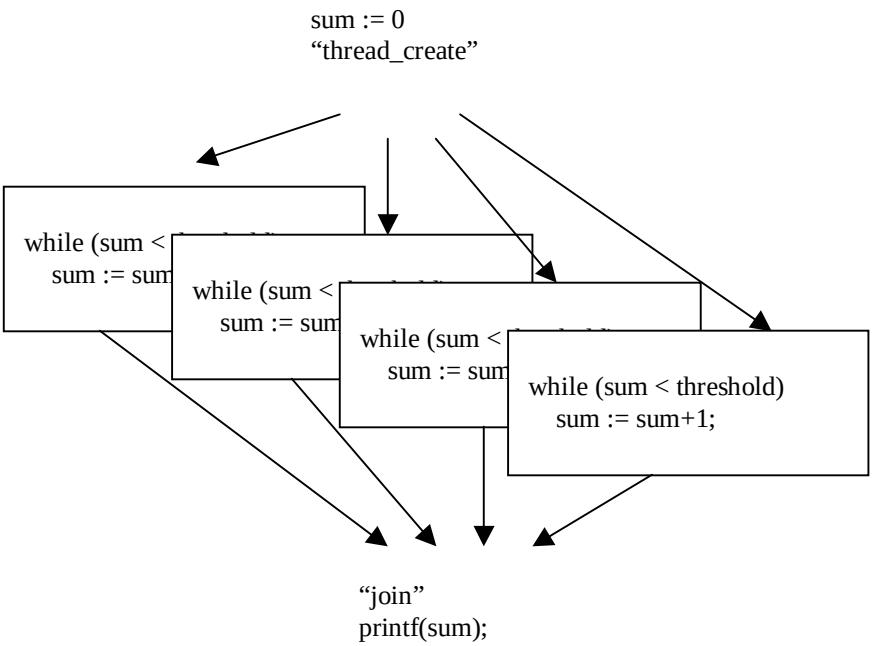
```
loop:   LD F0, 0(R1)          ;line 1
        LD F2, -8(R1)         ;line 2
        SUBI R1, R1, #8       ;line 5
        stall
        ADDD F4, F0, F2       ;line 3
        BNEZ R1, loop         ;line 6
        stall (or NOP)
        SD 8(R1), F4          ;line 4
```

d) Show how loop unrolling with maximum optimizations could help avoiding <u>all</u> the stalls in this loop using the smallest amount of unrolling. You can not introduce any new instruction types. You can only re-schedule, remove or re-offset existing ones. You do not have to show the set-up/clean-up code before and after the loop (4p)

```
loop:   LD F0, 0(R1)      ;line1 setup          F0 := x[i]
        LD F2, -8(R1)     ;line2                F2 := x[i+1]
        LD F6, -16(R1)    ;line 2'              F6 := x[i+2]
        SUBI R1, R1, #16  ;line 5 & line 5':    i := i+2
        ADDD F4, F0, F2   ;line 3:              F4 := x[i+1] + x[i]
        ADDD F8, F6, F2   ;line 3'              F8 := x[i+2]+x[i+1]
        BNEZ R1, loop     ;line 6 &line 6'      done yet?
        SD 16(R1), F4     ;line 4               x[i]:=F4
        SD 8(R1), F8      ;line 4'              x[i+1]:=F8
```

3. (Bonus 3) Synchronization and memory ordering (2+2+4=8p)

Consider the following parallel example:



Where each CPU performs:

```
while (sum < threshold)
      sum := sum + 1;
```

in the parallel section.

a) i) What is the highest and lowest number the printf(sum) may print?
Highest=threshold+3 Lowest=threshold
ii) What is the highest and lowest number of additions that may be performed?
Highest= 4 x threshold Lowest=threshold

b) Replace each parallel section with the following code:

```
lock(L)
while (sum < threshold)
      sum := sum + 1;
unlock(L)
```

i) What is the highest and lowest number the printf(sum) may print?
High=Low=threshold
ii) What is the highest and lowest number of additions that may be performed?
High=Low=threshold

c) Write pseudo code for an implementation of lock(L) and unlock(L) using the function test-and-set(L) that return the old value stored in the memory location L and atomically writes the value "1" to that memory location. The implementation should produce a reasonable amount of bus coherence traffic (i.e., no banging!) (4p)

```
lock( *L)
  while true {
    if (test_and_set(L) == 0)
       break;
    while (L!=0) {};
  }
/* this is an "optimistic" lock, as referred to in Question 8*/

unlock(*L)
  L := 0;
```

4. Random Questions (8p)

(At most two sentences per subquestion)

a) For embedded CPUs, a low power consumption is often essential. Write an expression for power consumption using the terms: voltage, frequency and area (i.e., capacitance)

Power = frequenzy * capacitance * voltage^2

b) Suggest two techniques for lowering power consumption

Turn off unused parts of the CPU; Run several slower parallel CPU cores at lower frequency and voltage.

c) Which (one or many…) of the programming paradigms OpenMP, MPI or Posix threads (ptherads) supports a coherent shared memory view?

OpenMP and pthreads

d) IA64 (as well as some other microprocessors) uses a technique to avoid branches for small if/then/else statements. Describe briefly how it works and what it is called.

Predication. The logical result of the *if* statement is saved in a cnd register. The *then* and *else* are both executed, byt their side-effecs are made conditional on the value of the cnd register

e) What is a queue-based lock and what is its advantage over spin locks?

The contenders are ordered first-come-first-served. Only the next in line will try to grab a free lock, which will create less traffic.

f) What do the acronyms UMA, NUMA and COMA stand for. What special application optimizations are needed for each one of the in order to achieve a good performance (two sentences each).

UMA=Uniform Memory Architecture. No specific optimization (apart from cache opt) is needed.
NUMA=Non-uniform Memory Architecture. Need to place execution and data to avoid remote accesses.
COMA=Cache-Only Memory Architecture. No specific placement optimization is needed.

g) Software-based distributed shared memory, as suggested by Kai Li, is an alternative way of cheaply achieving a shared-memory image between more loosely couple nodes. It relies on a commonly existing hardware/software support to do its coherence check (i.e., am I allowed to perform this operation?) with no extra access time penalty. How is this achieved?

The coherence unit is a page! The page access rights are used to emulate coherence state and the page handler will trap for coherence violations.

h) How are coherence check done in DSZOOM and what is the potential pros/cons of that technique compared with Li's?

In-line snippets are "compiled" into the code and perform the coherence checks at runtime. This will create extra overhead at execution time, but will allow for smaller coherence units and avoid false sharing.

5. Virtual Memory (2+2+2+2=8p)

(At most two sentences per subquestion)

The virtual memory system can be viewed as a "cache system":

a) what part of it corresponds to a cache line

A page

b) what is its associativity

Fully associative

c) what is its write strategy

Copy Back

d) what piece of hardware is used to determine if a requested item is in the "cache" or not

TLB

6. Optimize for Caches (2+2+4=8p)

Make trivial changes to the following C programs such that the cache will be better utilized.

a)

```
for (j = 0; j < N; j = j + 1)
              for (i = 0; i < N; i = i + 1)
                            x[i][j] = 2 * x[i][j];
```

Optimized:

```
for (i = 0; i < N; i = i + 1)
              for (j = 0; j < N; j = j + 1)
                            x[i][j] = 2 * x[i][j];
```

b)

```
for (i = 0; i < N; i = i + 1)
              for (j = 0; j < N; j = j + 1)
                            a[i][j] = 2 * b[i][j];

for (i = 0; i < N; i = i + 1)
              for (j = 0; j < N; j = j + 1)
                            c[i][j] = K * b[i][j] + d[i][j]/2
```

Optimized:

```
for (i = 0; i < N; i = i + 1)
              for (j = 0; j < N; j = j + 1)
                            a[i][j] = 2 * b[i][j];
                            c[i][j] = K * b[i][j] + d[i][j]/2
```

c) What kind of locality have you improved under a) and b) respectively? (4p)

Question a: spatial locality

Qusetion b: temporal locality

7. Random questions (8p)

a) Describe the different types of pipeline hazards: (2p)

i) structural hazards

several instruction, e.g., in different pipeline stages, need the same hardware resource at the same time.

ii) data hazards

e.g., RAW or WAR hazards. Instructions performed close in time can not produce/consume data "fast enough" to correctly interchange data.

b) Describe the two architectures: super-scalar and VLIW, and clearly state the difference between the two approaches (2p).

VLIW=each "instruction word" consists of a fixed number instructions performed in parallel. Explicit parallelism is found in the compiler. Super-scalar= The hardware detects independent instructions from a sequential "stream" of instructions and may launch them in parallel.

c) What is a victim cache and what kind of cache misses (Mark Hill's three Cs) does it remove (2p)?

A small, often very associative, cache that contains recently replaced cache lines. It is searched in parallel with the normal cache. Upon a hit in the VC, the VC cache line will be sent to the normal cache. This will effectively remove conflict misses.

d) IA64 calls it *register stack*, SPARC and the original RISC architecture call it *register window*. Describe briefly how one of these techniques work and what advantage it has over "normal" registers. (2p)

The register bank contains more registers than each instruction can logically name. The physical register name is calculated by adding an register offset to the logical register name. The register offset is increased/decreased for each *procedure call* in such a way that the calling and called procedure will share some physical registers (used for input/output parameters). Further, some physical registers are often common to all the procedures (aka, global registers). This scheme can avoid much of the overhead with saving and restoring of registers at procedure calls.

8. Cache coherence (10p)

All the three CPUs in a MOSI shared-memory multiprocessor executes the following code almost at the same time:

```
lock(L);        /* an optimistic spin lock with local spinning in the cache*/
A = A + 1;      /*is compiled to 3 instr: LD R1,#A; ADD R1,R1,#1; ST R1,#A*/
unlock(L);
<after a long time …>
<some other execution replaces A and L from the cache, if still present>
```

CPU1 is running slightly ahead of CPU2 and CPU3 and will get first to the critical section, but all of them will still have made their first attempt to acquire the lock L before CPU1 gets into the critical section. CPU2 is the second one to enter the critical section, and CPU3 is last.
The following four bus transaction types can be seen on the snooping bus connecting the CPUs:

- RTS: ReadtoShare (reading the data with the intention to read it)
- RTW, ReadToWrite (reading the data with the intention to modify it)
- WB: Writing data back to memory
- INV: Invalidating other caches copies

Initially, both A and L only reside in memory, both with a data value of zero.

Rip out and fill in the solution sheet at the end of the exam. Show every state change and/or value change of A and L in each CPU's cache according to a possible interleaving of the parallel memory instruction (including all memory instructions inside the lock and unlock routine). After the parallel execution is done for all of the CPUs, the cache lines still in the caches are being replaced. These actions should also be shown. For each line, also state what bus transaction occurs on the bus (if any) as well as which device is providing the corresponding data (if any).

Solution:

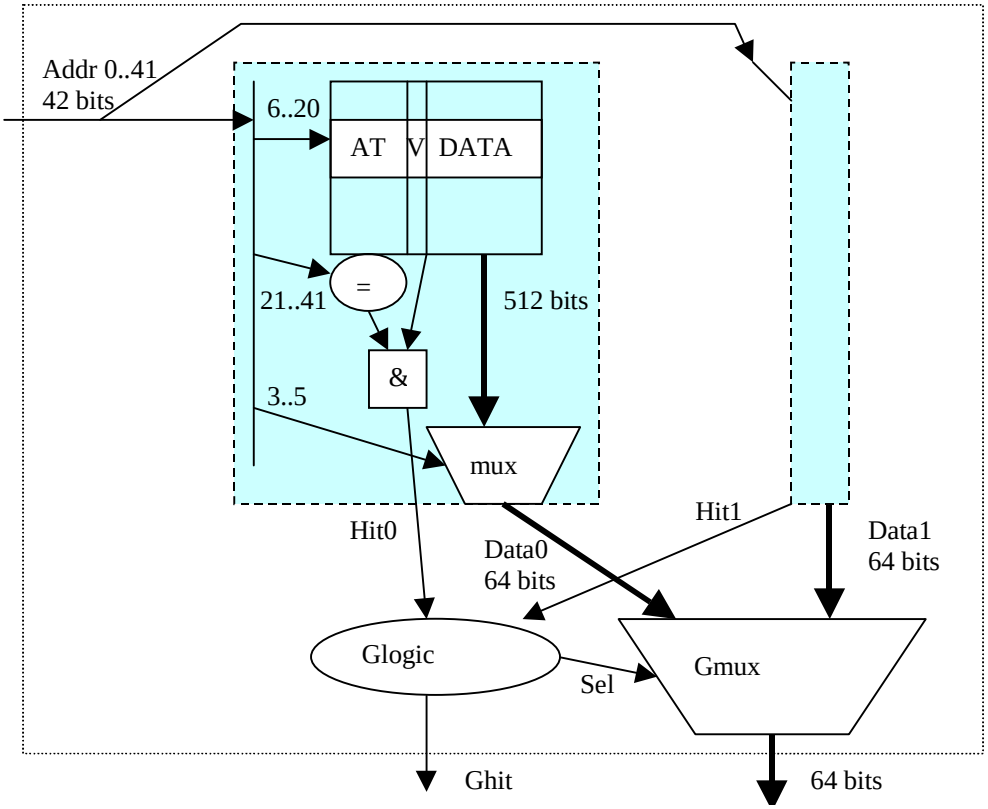
CPU action	**Bus transaction**	**State/value after the bus transaction**						**Data is provided by** [CPU 1, 2, 3 or Mem]
		CPU1		CPU2		CPU3		
		A	L	A	L	A	L	
Initially		I	I	I	I	I	I	
CPU1: TAS L	RTW(L)		M/1					Mem
CPU2: TAS L	RTW(L)		I		M/1			CPU1
CPU3: TAS L	RTW(L)				I		M/1	CPU2
CPU2: RD (while…)	RTS(L)				S/1		O/1	CPU3
CPU1: LD A	RTS(A)	S/0						Mem
CPU1: ST A	INV(A)	M/1						-
CPU1: ST L (unlock)	RTW(L)		M/0		I		I	CPU3
CPU2: LD L (while…)	RTS(L)		O/0		S/0			CPU1
CPU3: LD L (while…)	RTS(L)						S/0	CPU1
CPU2: TAS (lock)	INV(L)		I		M/1		I	-
CPU3: TAS (lock)	RTW(L)				I		M/1	CPU2
CPU2: LD A	RTS(A)	O/1		S/1				CPU1
CPU2: ST A	INV(A)	I		M/2				-
CPU2: ST L (unlock)	RTW(L)				M/0		I	CPU3
CPU3: LD L (while…)	RTS(L)				O/0		S/0	CPU2
CPU3: TAS L (lock)	INV(L)				I		M/1	-
CPU3: LD A	RTS(A)			O/2		S/2		CPU2
CPU3: ST A	INV(A)			I		M/3		-
CPU3: ST L (unlock)	-						M/0	-
… time elapses …	-							-
CPU3: replace A	WB(A)					I		CPU3
CPU3: replace L	WB(L)						I	CPU3

9. Caches (6+2 = 8p)

a) Make a drawing of a physical cache with the following data

- Cache size: 4Mbyte
- Cache line size: 64byte
- Organization: 2-way
- CPU word size: 8 byte (this is the size of the data unit delivered from the cache to the CPU)
- Physical address size: 42 bits (i.e., 4Tbyte of address space can be addressed)

Clearly show the address bits ranges used for the indexing, the comparisons, and multiplex (mux) selects, etc. Describe, either in words or in logic, the functionality of all logic needed to resolve a read access to the cache.(6p)



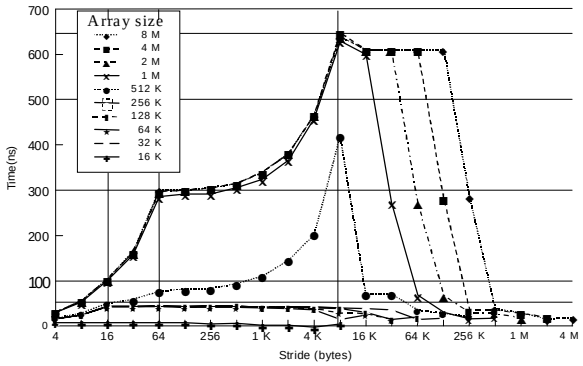
c) Which bit ranges would change if the cache size was increased to 8Mbyte while all the other parameters stayed the same? State the new bit ranges (you do not need to make completely new drawing) (2p)

Bit 21 from the address is added to the index range (i.e., now 6..21) and the address comparison is decreased by one bit (i.e., now 22..41).

10. Microbenchmarks (2+2+2+2=8p)

A simple microbenchmark code below is used to draw curves that characterizes the properties of the underlying memory system

```
for (times = 0; times < Max; time++) /* many times*/
  for (i=0; i < ArraySize; i = i + Stride)
    dummy = A[i]; /* touch an item in the array */
```



Next page contains four such diagrams (A, B, C and D).

a) One generation computer is used to produce graph A, the next generation product is used to produce graph B. What improvement does the new product contain?

The TLB is twice as large (and with a reach which is larger than the L2 cache's)

b) One computer system is characterized by the curve C. After updating its operating system its curve looks like B. What feature does the new operatingsystem implement?

Page coloring, i.e., it picks physical pages such that the lower bits of the page-frame address does not change in the V->P translation.

c) One generation computer is used to produce graph C, the next generation product is used to produce graph B. Both are running identical operating systems What improvement does the new cache hierarchy contain? (Clarification: B and C both have a two-level cache hierarchy)

The L2 cache of has become much more associative.

d) One generation computer is used to produce graph D, the next generation product is used to produce graph B. What improvement does the new product contain?

The TLB is much more associative (e.g., changing from direct mapped to fully associative)

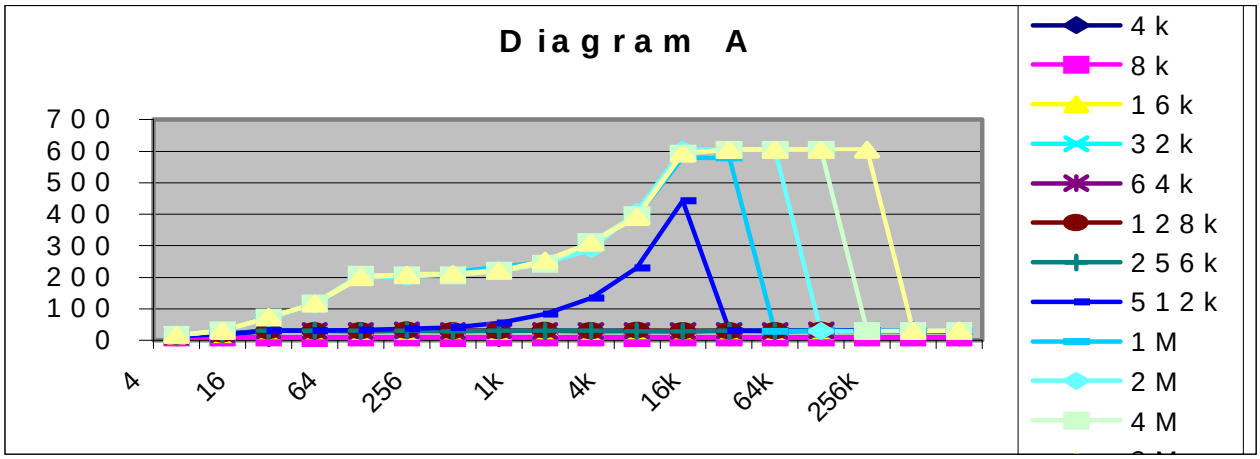


Diagram A
700
600
500
400
300
200
100
0
4
16
64
256
1k
4k
16k
64k
256k
4 k
8 k
16 k
32 k
64 k
128 k
256 k
512 k
1 M
2 M
4 M

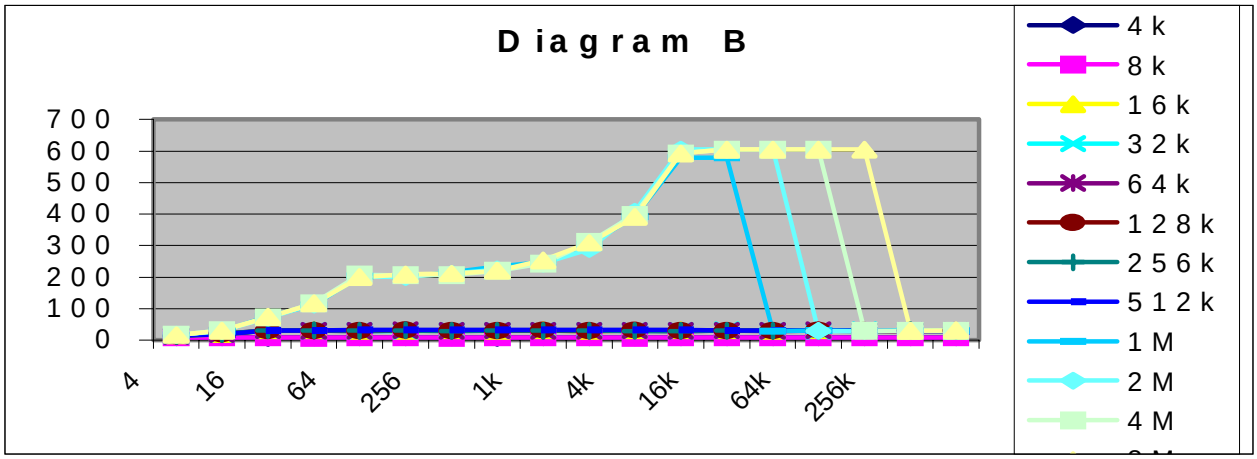


Diagram B
700
600
500
400
300
200
100
0
4
16
64
256
1k
4k
16k
64k
256k
4 k
8 k
16 k
32 k
64 k
128 k
256 k
512 k
1 M
2 M
4 M

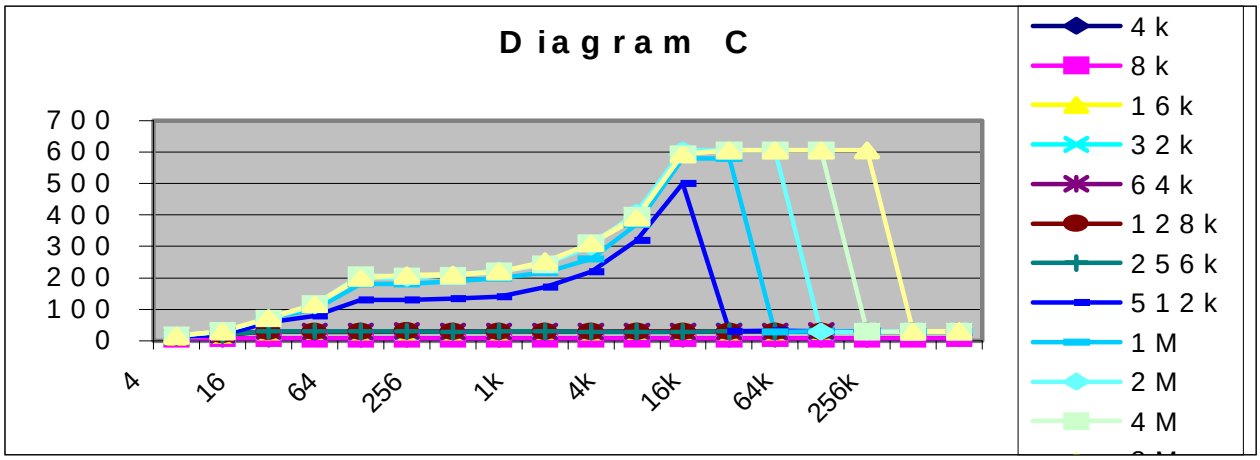


Diagram C
700
600
500
400
300
200
100
0
4
16
64
256
1k
4k
16k
64k
256k
4 k
8 k
16 k
32 k
64 k
128 k
256 k
512 k
1 M
2 M
4 M

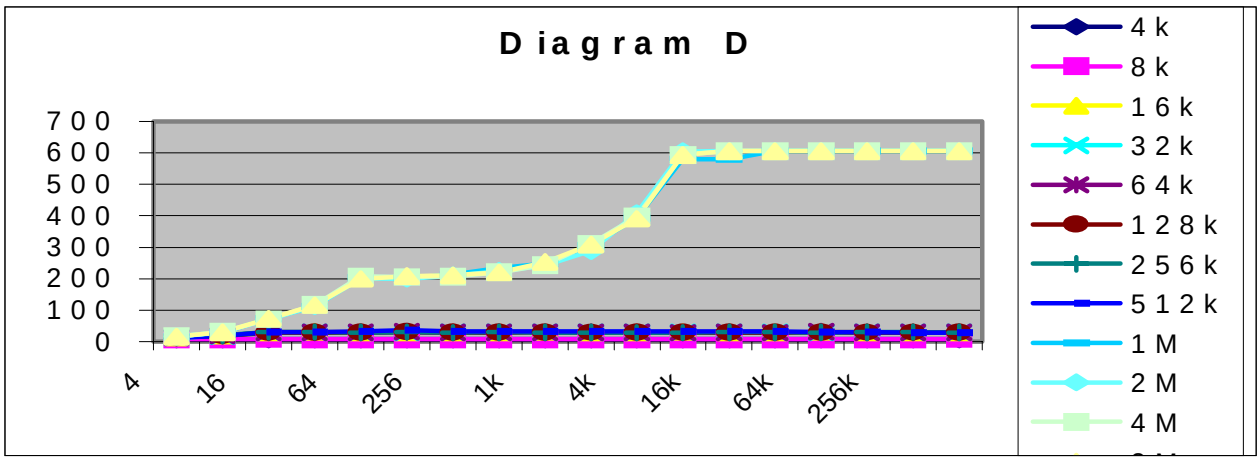


Diagram D
700
600
500
400
300
200
100
0
4
16
64
256
1k
4k
16k
64k
256k
4 k
8 k
16 k
32 k
64 k
128 k
256 k
512 k
1 M
2 M
4 M