Project 4.1 – Access the Memory Hierarchy using hwloc

yufanxues-mbp:hwloc-1.9 yufanxue$ lstopo

Machine (8192MB) + NUMANode L#0 (P#0 8192MB) + L3 L#0 (3072KB)

L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0

PU L#0 (P#0)

PU L#1 (P#1)

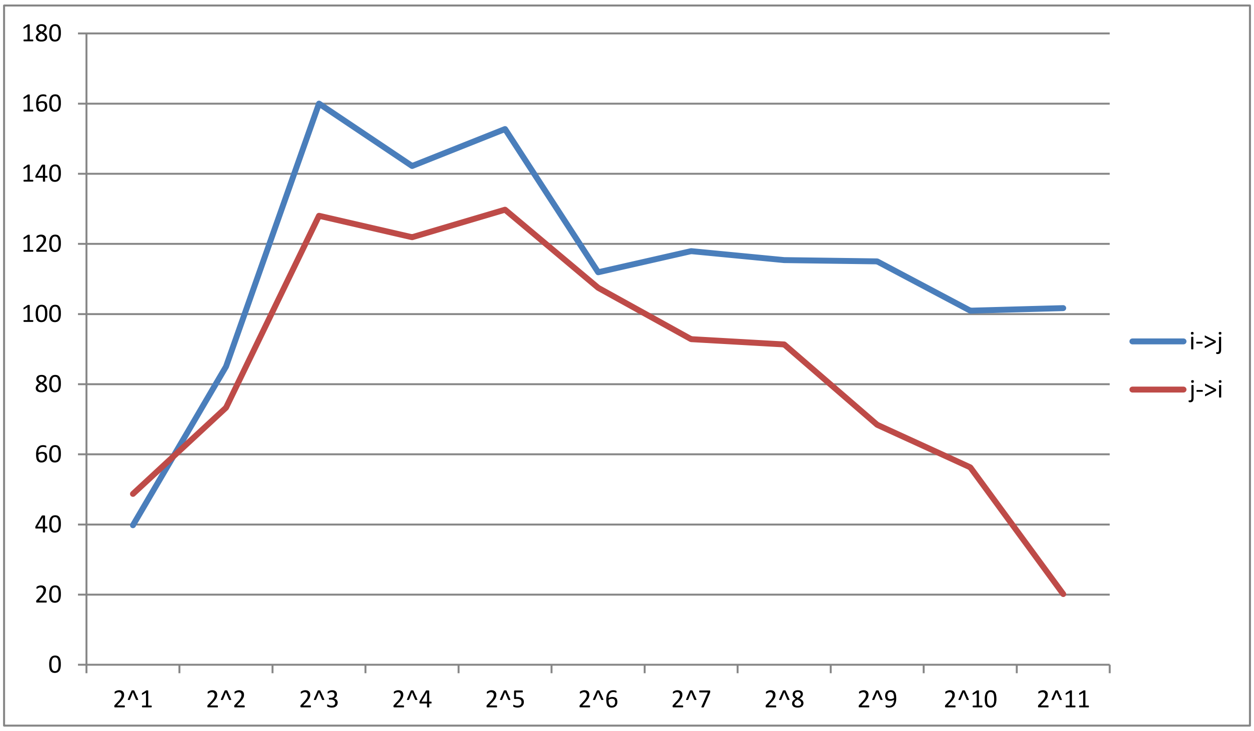
L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1

PU L#2 (P#2)

PU L#3 (P#3)

Project 4.2 – Access the Memory Hierarchy using hwloc

Part.1 Graph



Part.2 Data

[root@euca-192-168-132-153 code]# ./scalarmult 5000

2	32	229.779411994	48.699124682
4	128	85.046096110	73.303169153
8	512	160.001520174	128.013083065
16	2048	142.223660404	121.901459078
32	8192	152.737446734	129.742963861
64	32768	111.912640274	107.506967266
128	131072	117.947109090	92.827245471
256	524288	115.380821312	91.341309515
512	2097152	114.993867276	68.419718774
1024	8388608	100.973885979	56.290993546
2048	33554432	101.701408183	20.168067253

Part.3 Questions

Question 1: Which experiment produces the best overall performance

Answer: The i - > j is best, that's to say , the row and column is a better way to do matrix multiplication

Question 2: Describe the course of divergence of performance between the experiments

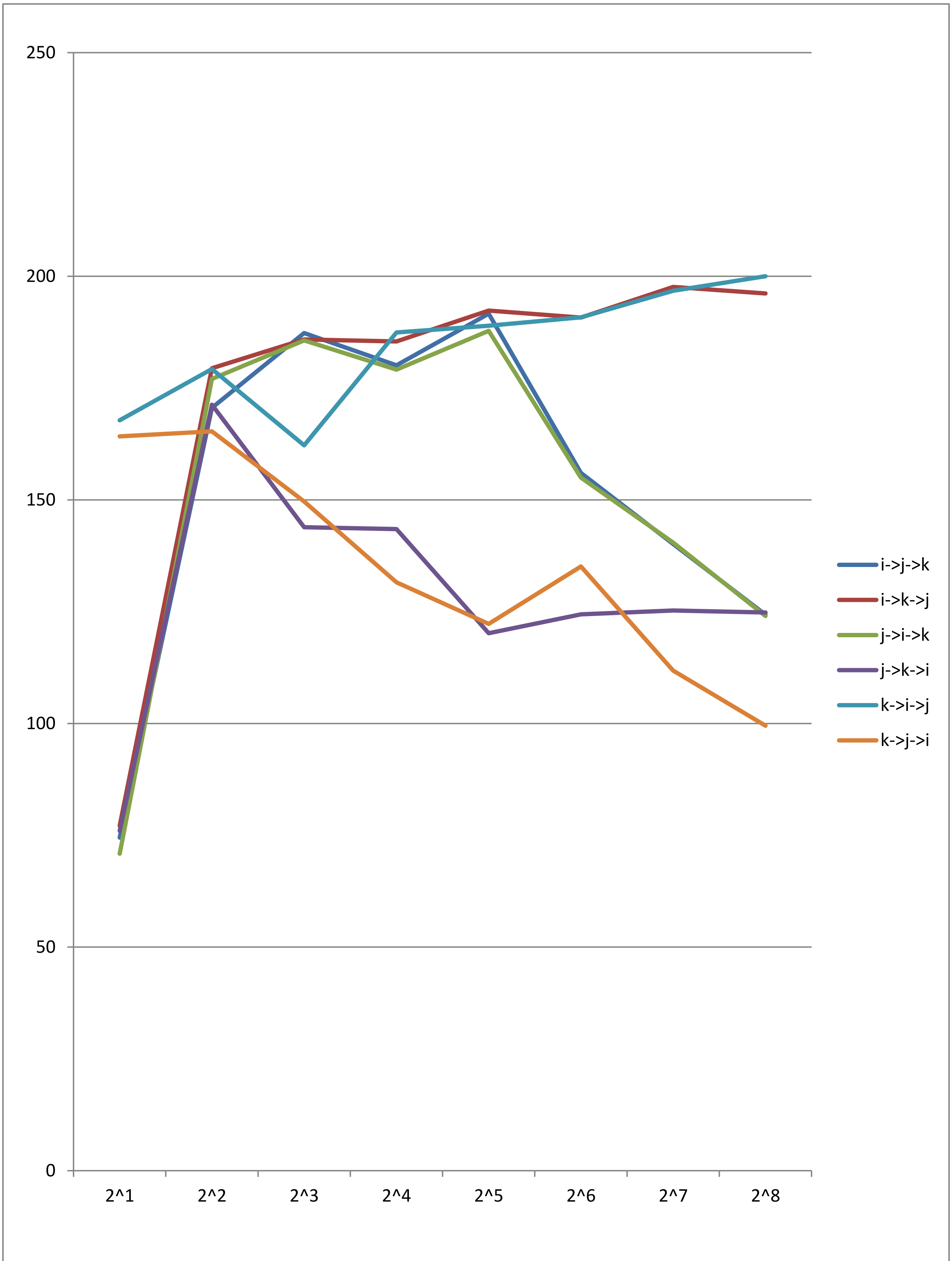
Answer: Since matrix is a 2-D array, and in fact, 2-D array was represented in memory by 1-D array, here, we suppose we have a 2-D array int[][] 2d_arr

For each row, it has A amount elements

It has N rows, so if we want get the element which position is [2][3], in memory, we have to calculate its offset with 1-D array, the way is offset = 2 * A + 3, by which means if we scan the 2-D array by the way j->i, in the inner loop, since j was variable for this inner loop, we have to convert the 2-D position into 1-D offset, however if we can the 2-D array by i->j, that means we will scan the memory in sequence, and we do not have to calculate every time.

Project 4.3 – Matrix Multiplication

Part.1 Graph


250
200
150
100
50
0
2^1
2^2
2^3
2^4
2^5
2^6
2^7
2^8
i->j->k
i->k->j
j->i->k
j->k->i
k->i->j
k->j->i

Part.2 Data

	i->j->k	i->k->j	j->i->k	j->k->i	k->i->j	k->j->i
2^1	74.4749286	77.12936096	70.8858831	75.97297649	167.785411	164.1945132
2^2	170.583695	179.4387326	177.0528939	171.2880313	179.2525785	165.3304182
2^3	187.294888	185.8754652	185.6662536	143.8995576	162.193929	149.6396149
2^4	180.099158	185.4429576	179.1368239	143.4833516	187.4424657	131.5807524
2^5	191.632422	192.3281998	187.7892073	120.1929066	188.9456232	122.2660934
2^6	156.028264	190.7685184	154.9941985	124.4152789	190.8199464	135.1033045
2^7	140.125231	197.6147377	140.4561294	125.2715004	196.7679731	111.8128971
2^8	124.317672	196.1739856	124.0474979	124.8310219	200.0103109	99.50097963

Part.3 Questions

Question 1: Which experiment produces the best overall performance? Why?

Answer: i->k->j and k->i->j produces the best overall performance

Since the megaFLOPS equation was

FLOPS = (2 * size * size * size) / time

With the same size, the less the FLOPS is, the more time it use, by this way , i->k->j and k->i->j always get the highest FLOPS that means it always use less time and perform better.

Question 2: Rank the six experiments from best to worst in terms of MFLOPS

1) i->k->j
2) k->i->j
3) i->j->k

4) j->i->k

5) j->k->i

6) k->j->i

Question 3: Describe the cause of the divergence of performance between the experiments.

From the equation:

matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];

If we make the loop follow this sequence that first the outer loop was i then k then j, that's to say, matrix_a and matrix_b we access in sequence, although matrix_c we didn't access in sequence, however, we didn't read data from matrix_c.

For 1) i->k->j, it made

access matrix_c in sequence (outer loop i inner loop j)

access matrix_a in sequence,(outer loop i inner loop k)

access matrix_b in sequence (outer loop k inner loop j)

For 2) k->i->j, it made

access matrix_c in sequence (outer loop i inner loop j)

access matrix_b in sequence (outer loop k inner loop j)

For 3) i->j->k, it made

access matrix_c in sequence (outer loop i inner loop j)

access matrix_a in sequence,(outer loop i inner loop k)

For 4) j->i->k, it made

access matrix_a in sequence,(outer loop i inner loop k)

For 5) j->k->i, it made all not in sequence

For 6) k->j->i, it made

access matrix_b in sequence (outer loop k inner loop j)

Question 4: You may notice that some of the experiments pair up. Why does happen?

Since we describe each experiments upon, we notice that

1) i->k->j and 2) k->i->j pair up, since

For 1) i->k->j, it made

access matrix_c in sequence (outer loop i inner loop j)

access matrix_a in sequence,(outer loop i inner loop k)

access matrix_b in sequence (outer loop k inner loop j)

For 2) k->i->j, it made

access matrix_c in sequence (outer loop i inner loop j)

access matrix_b in sequence (outer loop k inner loop j)

We notice that both 1) and 2) access matrix_c and matrix_b in sequence

3) and 4) pair up since

For 3) i->j->k, it made

access matrix_c in sequence (outer loop i inner loop j) access matrix_a in sequence,(outer loop i inner loop k)

For 4) j->i->k, it made

access matrix_a in sequence,(outer loop i inner loop k)

Both 3) and 4) access matrix_a in sequence and whether it access matrix_c in sequence seems doesn’t matter.