Unrolling & Accumulating:

Case 1 :   Lonestar

Intel® XEON® X5680 Double ADD *

- Super Computer Name : Lone Star.
- Microarchitecture Code Name : Nehalem.
- Compute Node :   CPU   @ 3.33GHz
- Theoretical Limit: 1.00 cycles per issue
- Latency : 3 Cycles

	L = 1	L = 2	L = 3	L = 4	L = 5	L = 6
K = 1	6.200880	6.202073	6.195767	2.855432	2.858605	2.854486
K = 2	0	1.457416	0	1.440694	0	1.440222
K = 3	0	0	0.984905	0	0	0.967512
K = 4	0	0	0	0.971356	0	0
K = 5	0	0	0	0	0.975273	0
K = 6	0	0	0	0	0	0.970029

Fig : Loop Unrolling vs accumulator table.

Observations :   Clearly for K = 3 we get the Maximum performance.

Floating Point Add Instruction :

Latency INTEL XEON =   3 Cycles,   Cycles per Issue = 1.

So,Latency/(Cycles per issue) = 3/1 = 3.

Therefore we have got 3 latency cycles per issue.

Comparison : The peak occurs at k = 3 and L = 6, not at k = 3 and L = 3.

(b) : best values of K : 3 ;
best values of L : 6 ;

As we increase the accumulators, we are inherently decreasing the inter dependency in the instructions.

Thus performance get closer to the execution throughput of the execution unit.
Now we run the scalaradd on compute node :

	C.P.I	I.P.C	MFlops/S	LOG(N)
N =200	2.282132	0.438187	1459.162	2.30103
N =400	2.162444	0.46244	1539.924	2.60206
N =600	2.10155	0.475839	1584.545	2.778151
N =1100	2.10182	0.475778	1584.341	3.041393
N =1900	2.100548	0.476066	1585.301	3.278754
N =3500	2.090138	0.478437	1593.196	3.544068
N =6200	1.39357	0.717581	2389.546	3.792392
N =11100	1.390853	0.718983	2394.214	4.045323
N =19900	1.448341	0.690445	2299.182	4.298853
N =35800	1.463812	0.683148	2274.882	4.553883
N =64300	1.510515	0.662026	2204.546	4.808211
N =115700	1.514042	0.660484	2199.411	5.063333
N =208300	1.514358	0.660346	2198.952	5.318689
N =374900	1.532789	0.652406	2172.51	5.573915
N =674700	1.532335	0.652599	2173.154	5.829111
N =1214400	1.677207	0.596229	1985.444	6.084362
N =2186000	2.519954	0.396833	1321.453	6.33965
N =3934700	2.773804	0.360516	1200.517	6.594912
N =7082400	2.768918	0.361152	1202.636	6.85018
N =12748300	2.773394	0.360569	1200.695	7.105452
N =22946900	2.773763	0.360521	1200.535	7.360724

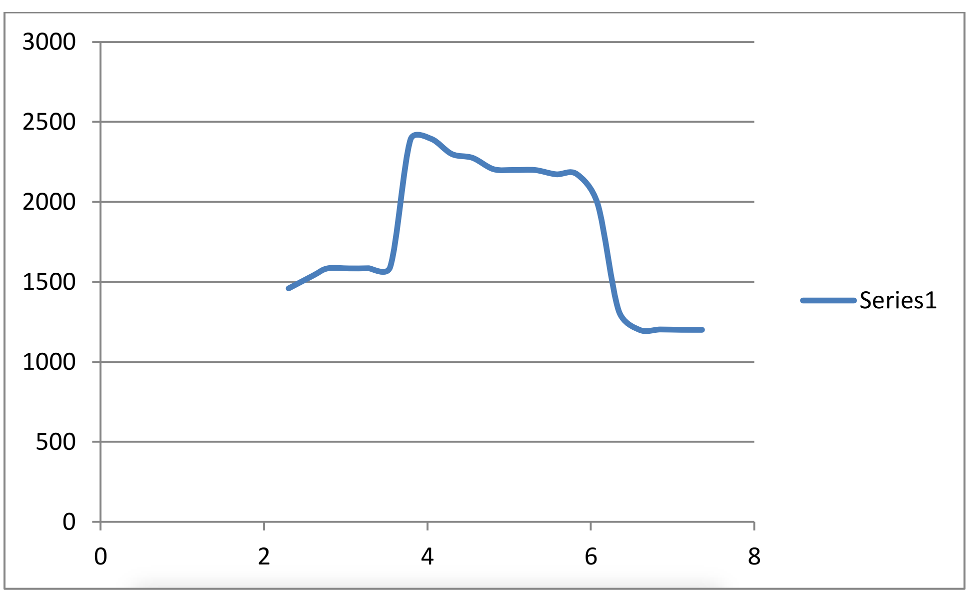

Figure : Semilog-plot, where X axis is log(N) and Y axis is MFlops/Seconds.

Lone star Architectural details :

First Level Cache size(L1) : 32 KB
Second Level Cache size(L2) : 256 KB
Third Level Cache size(L3) : 12288 KB

Clock frequency for lonestar : 3.33GHz.
Therefore time to for one clock cycle = (1/3.33) nano second = 0.33 nanoseconds.

Thus Latency is : 3 * .33 nano sec = 1 nanoseconds.
And throughput is : 0.33 nanoseconds.

Explanation : Till input size 1100 to 3500 the performance is almost same. But for input size 6200 and there onwards we see almost a continuous degradation in performance. The performance is very unexcpected and can't be explained on the basis of cache misses.

The code was then run with valgrind.

```
c340-105$ valgrind scalaradd out
==28821== HEAP SUMMARY:
==28821==     in use at exit: 568 bytes in 1 blocks
==28821==   total heap usage: 22 allocs, 21 frees, 413,050,968 bytes allocated
==28821==
==28821== LEAK SUMMARY:
==28821==    definitely lost: 0 bytes in 0 blocks
==28821==    indirectly lost: 0 bytes in 0 blocks
==28821==      possibly lost: 0 bytes in 0 blocks
==28821==    still reachable: 568 bytes in 1 blocks
==28821==         suppressed: 0 bytes in 0 blocks
==28821== Rerun with --leak-check=full to see details of leaked memory
==28821==
==28821== For counts of detected and suppressed errors, rerun with: -v
==28821== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```

Now using cg_annotate :

```
c340-105$ cg_annotate cachegrind.out.28771
--------------------------------------------------------------------------------
I1 cache:         67108864 B, 64 B, 2-way associative
D1 cache:         67108864 B, 64 B, 2-way associative
L2 cache:         268435456 B, 64 B, 8-way associative
Command:          scalaradd
```

```
Data file:        cachegrind.out.28771
Events recorded:  Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw
Events shown:     Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw
Event sort order: Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw
Thresholds:       99 0 0 0 0 0 0 0 0
Include dirs:
User annotated:
Auto-annotation:  off

--------------------------------------------------------------------------------
     Ir I1mr I2mr     Dr D1mr D2mr     Dw D1mw D2mw
--------------------------------------------------------------------------------
117,440  644  644 32,251  710  709 12,525  242  242  PROGRAM TOTALS

--------------------------------------------------------------------------------
    Ir I1mr I2mr     Dr D1mr D2mr    Dw D1mw D2mw  file:function
--------------------------------------------------------------------------------
```

Case 2 : Stampede

- Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
- cpu MHz   : 2701.000
- Microarchitecture Code Name : Sandy Bridge
- Theoretical Limit: 1.00 cycles per issue
- Latency : 3 Cycles

L1 cache : 32KB(Instruction Cache) + 32 KB(data cache)
L2 data cache : 256 KB
L3 cache : 20480 KB

Stampede Results using perf tool :

```
c557-402$ perf stat ./scalaradd out

 Performance counter stats for './scalaradd out':

       2335.015514 task-clock               #    0.999 CPUs utili
                20 context-switches         #    0.000 M/sec
                 4 CPU-migrations           #    0.000 M/sec
               162 page-faults              #    0.000 M/sec
     7,241,829,485 cycles                   #    3.101 GHz
     3,628,614,451 stalled-cycles-frontend  #   50.11% frontend c
```

1,312,437,849 stalled-cycles-backend   #   18.12% backend c
9,007,924,570 instructions   #   1.24 insns per
                                        #   0.40 stalled cy
1,357,841,303 branches   # 581.513 M/sec
    2,486,026 branch-misses   #   0.18% of all bra
  2.338124946 seconds time elapsed

Stampede Results :

	L = 1	L = 2	L = 3	L = 4	L = 5	L = 6
K = 1	2.603488	2.584679	2.584202	2.583229	2.582863	2.581009
K = 2	0	1.304106	0	1.302441	0	1.300713
K = 3	0	0	0.885721	0	0	0.885838
K = 4	0	0	0	0.886562	0	0
K = 5	0	0	0	0	0.886638	0
K = 6	0	0	0	0	0	0.887187

Fig : Loop Unrolling vs accumulator table.

Observations : Clearly for K = 3 and L =3, we get the Maximum performance.

	C.P.I	I.P.C	GFlops/S	LOG(N)
N =200	0.893588	1.119084	3.021527	2.30103
N =400	0.889374	1.124386	3.035843	2.60206
N =600	0.888018	1.126103	3.040479	2.778151
N =1100	0.886335	1.128242	3.046252	3.041393
N =1900	0.885626	1.129145	3.048691	3.278754
N =3500	0.885199	1.129689	3.050162	3.544068
N =6200	0.887197	1.127145	3.043293	3.792392
N =11100	0.888920	1.124961	3.037394	4.045323
N =19900	0.887230	1.127103	3.043179	4.298853
N =35800	1.029795	0.971067	2.621881	4.553883
N =64300	1.146938	0.871887	2.354094	4.808211
N =115700	1.147812	0.871223	2.352302	5.063333
N =208300	1.147774	0.871252	2.352379	5.318689
N =374900	1.169493	0.855071	2.308693	5.573915
N =674700	1.169053	0.855393	2.309562	5.829111
N =1214400	1.169229	0.855264	2.309214	6.084362
N =2186000	1.375981	0.726754	1.962236	6.33965
N =3934700	1.930751	0.517933	1.39842	6.594912

N =7082400	1.968611	0.507972	1.371525	6.85018
N =12748300	1.968106	0.508103	1.371877	7.105452
N =22946900	1.968264	0.508062	1.371767	7.360724

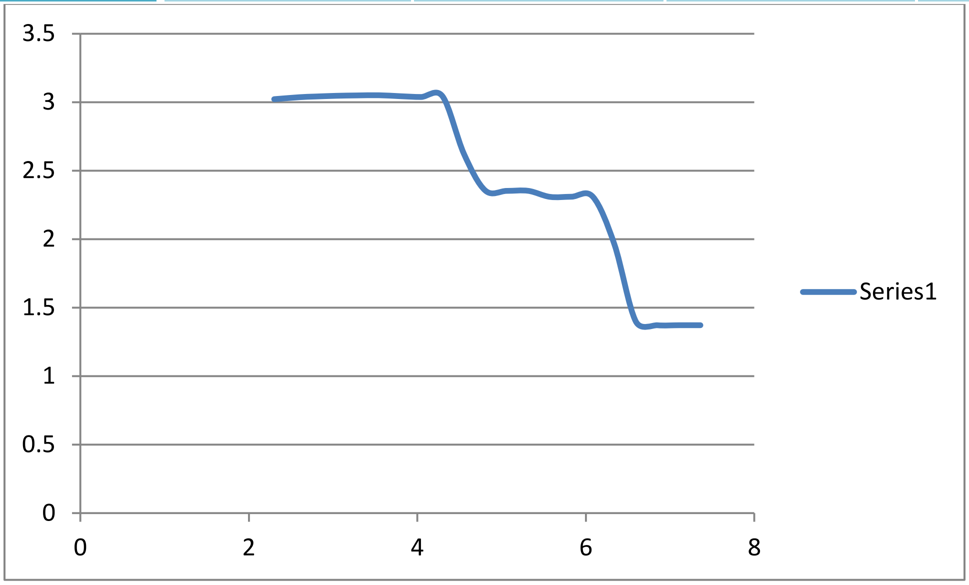

Figure : Semilog-plot, where X axis is log(N) and Y axis is MFlops/Seconds.

We see three plateau's :

First plateau corresponds to N = 200 to 19900
We ran perf for for I=1 to 9 and got output as :

```
c557-604$  perf stat -e cycles -e instructions -e L1-dcache-loads -e L1-dcache-load-misses ./scalaradd out
Performance counter stats for './scalaradd out':

    7,262,818,359 cycles                #    0.000 GHz
   16,358,447,690 instructions          #    2.25  insns per cycle
    4,064,592,946 L1-dcache-loads
      214,830,924 L1-dcache-load-misses #    5.29% of all L1-dcache hits

      2.363464778 seconds time elapsed
```

///

Second plateau corresponds to N = 38800 to N = 1214400

Perf Results for I=10 to16

```
c557-604$  perf stat -e cycles -e instructions -e L1-dcache-loads -e L1-dcache-load-misses ./scalaradd out
Performance counter stats for './scalaradd out':
```

```
   6,962,896,656 cycles                 #    0.000 GHz
  14,027,599,667 instructions           #    2.01  insns per cycle
   3,538,454,365 L1-dcache-loads
     442,647,052 L1-dcache-load-misses     #   12.51% of all L1-dcache hits

       2.248642886 seconds time elapsed
```

//

Third plateau corresponds to N = 2186000 to N = 22946900

Perf Results for I=17 to21

```
c557-604$  perf stat -e cycles -e instructions -e L1-dcache-loads -e L1-dcache-load-misses ./scalaradd out
 Performance counter stats for './scalaradd out':

   8,134,560,828 cycles                 #    0.000 GHz
   9,905,667,442 instructions           #    1.22  insns per cycle
   3,044,891,255 L1-dcache-loads
     379,590,383 L1-dcache-load-misses     #   12.47% of all L1-dcache hits

       2.627048102 seconds time elapsed
```

//

Perf overall run for I = 1 to 21

```
Performance counter stats for './scalaradd out':

  28,157,084,114 cycles                 #    0.000 GHz
  41,537,886,218 instructions           #    1.48  insns per cycle
  10,980,160,810 L1-dcache-loads
   1,134,977,904 L1-dcache-load-misses     #   10.34% of all L1-dcache hits

       8.459961283 seconds time elapsed
```

Conclusion :

Between N = 200 to 19900 the L1-dcache-load-misses   is 5.29% and 2.25 insns per cycle
Between N = 38800 to 1214400 the L1-dcache-load-misses   is 12.51% and 2.01 insns per cycle
Between N = 2186000 to 22946900 the L1-dcache-load-misses   is 12.47% and1.22 insns per cycle
Thus as N increases, I.P.C is decreasing and cache misses are increasing. Hence, we see decrease in performance.