Carter Levin
Assignment 1
CPSC 424

Loaded modules: Langs/Intel/15
Development Environment: Sublime SFTP
Commands to build:

```
$ sh build.sh
$ qsub runPiBench.sh
```

Env :

```
MKLROOT=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl
MANPATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/man/en_US:/
home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel64/shar
e/man/:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel
64_mic/share/man/:/usr/share/man:/opt/moab/share/man:
GDB_HOST=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gd
b/intel64_mic/bin/gdb-ia-mic
HOSTNAME=compute-33-1.local
IPPROOT=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp
INTEL_LICENSE_FILE=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/li
censes:/opt/intel/licenses:/home/apps/fas/Licenses/intel_site.lic
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
GDBSERVER_MIC=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debu
gger/gdb/target/mic/bin/gdbserver
SSH_CLIENT=10.191.63.252 54546 22
LIBRARY_PATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/../co
mpiler/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/lib/int
el64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/compiler/lib/intel64:/h
ome/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/lib/intel64:/home/apps/f
as/Langs/Intel/2015_update2/composer_xe_2015.2.164/tbb/lib/intel64/gcc4.4
PERL5LIB=/opt/moab/lib/perl5
FPATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/include
QTDIR=/usr/lib64/qt-3.3
OLDPWD=/home/fas/cpsc424/cel49
QTINC=/usr/lib64/qt-3.3/include
MIC_LD_LIBRARY_PATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.16
4/mpirt/lib/mic:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/lib/mic:/
home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/compiler/lib/mic:/home/app
s/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/lib/mic:/opt/intel/mic/coi/device-li
nux-release/lib:/opt/intel/mic/myo/lib:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2
015.2.164/tbb/lib/mic
```

```
SSH_TTY=/dev/pts/10
ANT_HOME=/opt/rocks
USER=cel49
LD_LIBRARY_PATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mpi
rt/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/../compiler
/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/lib/intel64:/h
ome/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/tools/intel64/perfsys:/opt
/intel/mic/coi/host-linux-release/lib:/opt/intel/mic/myo/lib:/home/apps/fas/Langs/Intel/2015_updat
e2/composer_xe_2015.2.164/compiler/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/co
mposer_xe_2015.2.164/mkl/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe
_2015.2.164/tbb/lib/intel64/gcc4.4:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_201
5.2.164/debugger/ipt/intel64/lib
MIC_LIBRARY_PATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/co
mpiler/lib/mic:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mpirt/lib/mic:
/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/tbb/lib/mic
ROCKS_ROOT=/opt/rocks
CPATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/include:/hom
e/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/include:/home/apps/fas/La
ngs/Intel/2015_update2/composer_xe_2015.2.164/tbb/include
YHPC_COMPILER=Intel
NLSPATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/compiler/lib/int
el64/locale/%l_%t/%N:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/
lib/intel64/locale/%l_%t/%N:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.16
4/mkl/lib/intel64/locale/%l_%t/%N:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_201
5.2.164/debugger/gdb/intel64_mic/share/locale/%l_%t/%N:/home/apps/fas/Langs/Intel/2015_up
date2/composer_xe_2015.2.164/debugger/gdb/intel64/share/locale/%l_%t/%N
MAIL=/var/spool/mail/cel49
PATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/bin/intel64:/home/a
pps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mpirt/bin/intel64:/home/apps/fas/L
angs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel64_mic/bin:/home/apps/f
as/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel64/bin:/home/apps/f
as/Modules:/usr/lib64/qt-3.3/bin:/opt/moab/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:
/sbin:/usr/java/latest/bin:/opt/rocks/bin:/opt/rocks/sbin:/home/fas/cpsc424/cel49/bin:/home/apps/
bin
YHPC_COMPILER_MINOR=164
TBBROOT=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/tbb
F90=ifort
PWD=/home/fas/cpsc424/cel49/cel49_ps1_cpsc424
_LMFILES_=/home/apps/fas/Modules/Base/yale_hpc:/home/apps/fas/Modules/Langs/Intel/15
YHPC_COMPILER_MAJOR=2
JAVA_HOME=/usr/java/latest
GDB_CROSS=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/
gdb/intel64_mic/bin/gdb-mic
```

```
DOMAIN=omega
LANG=en_US.iso885915
MODULEPATH=/home/apps/fas/Modules
MOABHOMEDIR=/opt/moab
YHPC_COMPILER_RELEASE=2015
LOADEDMODULES=Base/yale_hpc:Langs/Intel/15
KDEDIRS=/usr
F77=ifort
MPM_LAUNCHER=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debug
ger/mpm/bin/start_mpm.sh
CXX=icpc
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
HISTCONTROL=ignoredups
INTEL_PYTHONHOME=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/d
ebugger/python/intel64/
SHLVL=1
HOME=/home/fas/cpsc424/cel49
FC=ifort
LOGNAME=cel49
QTLIB=/usr/lib64/qt-3.3/lib
CVS_RSH=ssh
SSH_CONNECTION=10.191.63.252 54546 10.191.12.33 22
MODULESHOME=/usr/share/Modules
LESSOPEN=||/usr/bin/lesspipe.sh %s
arch=intel64
INFOPATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gd
b/intel64/share/info/:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debu
gger/gdb/intel64_mic/share/info/
CC=icc
INCLUDE=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/include
G_BROKEN_FILENAMES=1
BASH_FUNC_module()=() {  eval `/usr/bin/modulecmd bash $*`
}
```

(1) To benchmark the divide operation's performance, two loops were run N = 1000000000 times. One contained the floating point operations, add, multiply, and divide, and the other had all the same operations except for the divide. The time to complete each loop was recorded and the difference between the two calculated. Performance was then calculated in MFlops/s by computing N (Multiplication Flops) / difference (s). The latency of the divide function was also calculated using the time difference, d, between the two loops and the number of executions, N, of each loop. If N divide operations execute in d seconds and the processor runs at 2.8 GHz, we can derive the equation, CPI = (d / N) * 2.8 GHz which, because N = 10^9, simplifies to d * 2.8 Hz. The code was compiled using four different compiler options to test their effects on performance producing the following table.

Trial	**options**	**With divide (s)**	**Without divide (s)**	**Difference (s)**	**MFlops / s**	**Latency (CPI)**
a	**-g -O0 -fno-alias -std=c99**	7.187129	4.572638	2.614491	382.4836268	7.3205748
b	**. -g -O1 -fno-alias -std=c99**	7.188821	3.266679	3.919178	255.1555454	10.9736984
c	**-g -O3 -no-vec -no-simd -fno-alias -std=c99**	7.186383	1.485624	5.700759	175.4152386	15.9621252
d	**-g -O3 -xHost -fno-alias -std=c99**	7.186866	1.485912	5.700954	175.4092385	15.9626712
e	**-g -O3 -xHost -fno-alias -prec-div -std=c99**	7.185714	1.48454	5.701174	175.4024697	15.9632872

The effects of the different compiler options on the runtime can be interpreted a number of different ways. For trial 'a', without any compiler optimization flags, one can see that the runtimes for the two loops is the most similar, with only a difference of 2.6 seconds. This figure leads to a Latency calculation of only 7.3 CPI. This figure is likely not very accurate, without any optimization and prefetching the calculations might not be done in cpu registers. Since some of these memory operations can be done in parallel with the mathematical ops, and because the divide op has a large latency, it is possible that they are only creating a significant bottleneck in the second loop without the divide. This would therefore result in a false comparison between the runtime of memory ops and computational ops. This error is further indicated by the fact that the calculated divide latency increases with compiler optimization. This suggests, not that divide is actually getting slower with optimization, but rather that the optimization is distilling the runtime of the loops down to the runtime of the computations and not memory operations.

The distinction between the runtimes with the -O1 flag versus the -O3 flag also indicates an interesting phenomenon. The -O1 flag prohibits loop unrolling, which means that each iteration of the loop must wait for the previous iteration to finish in order to begin. The -O3 command, on the other hand enables it, which means that, because in both loops the point_sq calculation does not depend on the result of the volume calculation, the kernel can begin working on calculating the next value of point_sq before the current iteration of the loop is complete. This means that the latency of memory operations interferes even less when the -O3 option is present, therefore making the results from trials b and c potentially even more accurate. However, the -O3 command also enables pipelining which could also explain the . The program could have pipelined the point_sq operation because it does not depend on any previous calculation. To test this I changed the point_sq calculation to depend on the volume, changing the calculation from, *1.0 + i * i,* to, *1.0 + volume * i*. This change forces the kernel to complete the volume calculation before moving on to the next iteration of the loop and results in slightly different results. The difference between the -O1 results and the -O3 results decreases slightly, potentially indicating a reduction in pipeline optimization. A more pronounced difference however is the increased latency of the divide calculation. It is possible that, while the new program obviously no longer estimates pi, by forcing the loop to complete before executing the next iteration it disables certain optimizations and does a better job benchmarking the divide latency.

Trial	Options	With divide (s)	Without divide (s)	Difference (s)	MFlops / s	Latency (CPI)
a	**-g -O0 -fno-alias -std=c99**	13.720154	6.534866	7.185288	139.1732663	20.1188064
b	**-g -O1 -fno-alias -std=c99**	10.126178	2.939426	7.186752	139.1449155	20.1229056
c	**-g -O3 -no-vec -no-simd -fno-alias -std=c99**	9.147076	2.94151	6.205566	161.1456554	17.3755848
d	**-g -O3 -xHost -fno-alias -std=c99**	9.145216	2.939489	6.205727	161.1414746	17.3760356
e	**-g -O3 -xHost -fno-alias -prec-div -std=c99**	9.145039	2.940052	6.204987	161.1606922	17.3739636

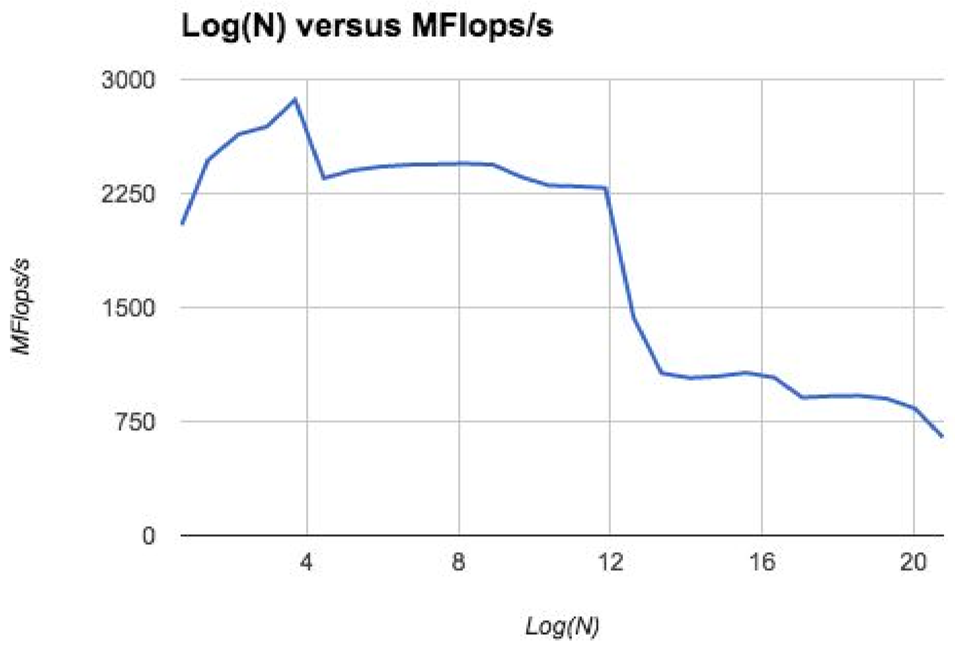
The similarities between trial c and d also indicate some of the programs properties. The fact that the -no-simd and -no-vec commands did not make a perceptible effect on runtime suggests that the program is not set up to take advantage of the nehalem processors multiple add and multiply computational units. This makes sense seeing as the operations “multiply add,” “divide”, and “add” all have their own arithmetic units.

The final trial, e, was run simply as a precaution in order to make sure the the divide operation was not being transformed into a multiply. Adding the -prec-div forces the compiler to perform the operations in the divide kernel. Because the result did not differ from the other tests run with the -O3 flag it’s likely that this was not an issue.

(2) The vector triad benchmark provided a graph with a number of distinct traits. Initially, as the size of N increases so does the number of MFlops per second. Between N = 2 and N = 19, the number of of MFlops/s successively increases from 2041 to 2866. This initial rise in MFlops can be attributed to the “wind up” phase in the vector triad kernel. The vector triad kernel breaks the multiply add function up into several steps. By splitting up, load, store, decode, and other steps into discrete levels, the hardware is able to pipe several add multiply operations in parallel. The depth of the pipe therefore results in latency when the number operations is small, but this latency becomes negligible after N increases beyond a certain length. After N exits the “wind up” phase the number of MFlops per second begins to level off. This plateau eventually gives way to three distinct breakdowns.

Each of these breakdowns signifies the point where N has grown too large for the current cache level, thus forcing the CPU to dropped down to the next tier. The first drop off signifies a move from L1 to L2 cache, the second from L2 to L3, and the third from L2 to main memory. Using the reported mflops at each plateau the bandwidth of the different memory tiers can be estimated. Since the floating point operation, a[i] = b[i] + c[i] * d[i], requires three loads and one store, each of size double, the number of bytes per flop is 4 * 8 = 32 bytes / flop. Thus at the first plateau where GFlops average out at 2.8 per second, the bandwidth for the L1 cache can be calculated to be 2.8 GFlops/s * 32 GB / GFlop = 89.6 GB / GFlop. Using this same logic the following table can be produced.

	GFlops / s	GB / GFlop	GB / s
Cache 1	2.8	32	89.6
Cache 2	2.4	32	76.8
Cache 3	1	32	32
Main Memory	0.65	32	20.8



Log(N)	MFlops/s
0.693147	2041.115837
1.386294	2469.392833
2.197225	2639.284283
2.944439	2689.79059
3.688879	2866.285582
4.442651	2350.355011
5.192957	2401.700055
5.934894	2425.803826
6.677083	2438.847339
7.418781	2442.193665
8.16109	2446.980188
8.903136	2439.81638
9.64517	2359.466357
10.387117	2302.498038
11.129055	2296.808032
11.870991	2286.591448

12.612933	1435.492742
13.354871	1067.232329
14.096809	1036.594644
14.838747	1046.56161
15.580684	1070.536584
16.322622	1038.956591
17.064559	907.701941
17.806496	916.769264
18.548434	919.82411
19.290371	900.258684
20.032308	836.300184
20.774246	645.63821