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/share/man/:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel64_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/gdb/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/licenses:/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/debugger/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/../compiler/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/ipp/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/compiler/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/lib/intel64:/home/apps/fas/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_2015.2.164/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/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/lib/mic:/opt/intel/mic/coi/device-linux-release/lib:/opt/intel/mic/myo/lib:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/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/mpirt/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:/home/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_update2/composer_xe_2015.2.164/compiler/lib/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_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_2015.2.164/debugger/ipt/intel64/lib
MIC_LIBRARY_PATH=/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/compiler/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:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mkl/include:/home/apps/fas/Langs/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/intel64/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_2015.2.164/mkl/lib/intel64/locale/%l_%t/%N:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel64_mic/share/locale/%l_%t/%N:/home/apps/fas/Langs/Intel/2015_update2/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/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/mpirt/bin/intel64:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel64_mic/bin:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/gdb/intel64/bin:/home/apps/fas/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/debugger/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/debugger/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/gdb/intel64/share/info/:/home/apps/fas/Langs/Intel/2015_update2/composer_xe_2015.2.164/debugger/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.

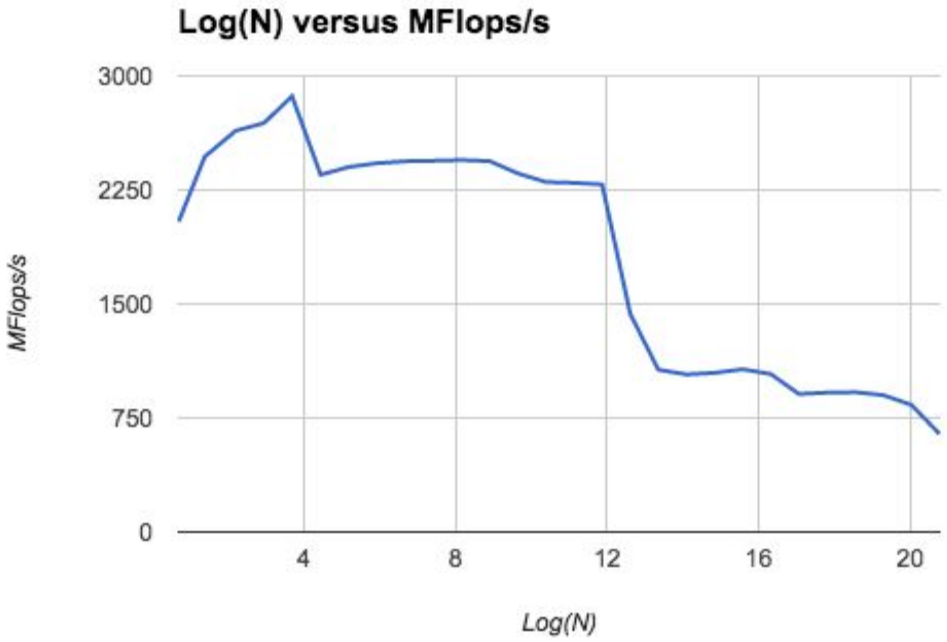
options	**With divide (s)**	**Without divide (s)**	**Difference (s)**	**MFlops / s**	**Latency (CPI)**
-g -O0 -fno-alias -std=c99	7.187129	4.572638	2.614491	382.4836268	7.3205748
. -g -O1 -fno-alias -std=c99	7.188821	3.266679	3.919178	255.1555454	10.9736984
-g -O3 -no-vec -no-simd -fno-alias -std=c99	7.186383	1.485624	5.700759	175.4152386	15.9621252
-g -O3 -xHost -fno-alias -std=c99	7.186866	1.485912	5.700954	175.4092385	15.9626712

In this table we can see a distinct difference between the runtime of the loops without the divide op., when the optimized flags are and are not used. The speedup is likely due to the use of "pipelining" in the kernel. This also may be related to the ways in which the variables are being fetched. When there is no optimization, or "pre-fetching," for instance the calculations may not be taking place in strictly in cpu registers. Thus the latency inate in memory operations may be clouding results. When the optimization flags are on the CPU is able to pre load and store in parallel, thus not affecting the results. It's also possible that the loop with divide was affected by the optimization, but the latency of the divide operation made the effect unnoticeable. The results also could have been influenced by parallelization in the intel nehalem processor. The processor has several floating point addition and multiplication units but only one divide operator. This means it can perform a multiply and add in parallel but only one divide at a time. Furthermore the pipeline is much deeper for the multiply operator than it is for the mult/add. This is perhaps why the loop without the divide, was able to be sped up so much by optimization.

(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