Proposal for SC22 Intramural Competition of Southeast University

[redacted]

0. Environment

Type	Specifications
Hardware	Apple MacBook Pro 2021 14"
	M1 Pro (8 cores, 6P2E)
	32G Unified Memory, 512G Internal SSD
Software	System: macOS 12.3.1 (21E258)
	Toolchain: Xcode 13.3 (13E113)

Since Apple doesn't provide technical specifications, the theoretical peak performance is unknown.

1. HPL

1.1 Environment

Type	Specifications
Compiler	`mpicc-openmpi-mp` (`Apple clang version 13.1.6 (clang-1316.0.21.2)`)
Math Library	Apple Accelerate framework, or the `vecLib` if specifically
MPI	Open MPI 4.1.3 from MacPorts

```
mpirun-openmpi-mp -np 6 ./xhpl | tee HPL.out
```

1.2 Optimization

1.2.1 Math Library

Actually, the `MacOSX_Accelerate` is a template provided with the HPL software package, so I first tried the `vecLib` by Apple and it turned out to perform well.

Later I also tried OpenBLAS. Although according to other test results of Apple M1 (154 GFLOPS), there should not be much difference with `vecLib`. Yet I got I much lower result of about 0.1x (22.61 Gflops) and currently have no idea why.

Intel MKL is not tested as it is already report to perform badly on Apple Silicon (which MATLAB uses). Also it currently not (or maybe will never) provide an arm64 build, which will certainly perform worse if run with Rosetta.

For GPU, I didn't find a Metal-based BLAS library. As for OpenCL, there exists a `CLBlast`. Although I successfully built HPL with CLBlast as dynamic library, I got a runtime error `libc++abi: terminating with uncaught exception of type clblast::CLCudaAPIError: OpenCL error: clGetMemObjectInfo: -38` and have no idea how to fix.

1.2.2 Tuning HPL.dat

Roughly calculated via HPL Calculator.

P × Q is usually considered to match the CPU cores, but as for ARM big.LITTLE architecture, it doesn't seem to be the best choice. According to my tests, setting P × Q to 6 (performance cores) instead of 8 (performance + efficiency cores) will lead to a better result, which is ~10 Gflops more.

NB is not expected to be too large, and 224 seems to be a commonly used value. I also ran a few tests on small scales (N = ~500), which proves 224 is a reasonable value.

As we got NB, the calculator will help with an aligned N value. Though my laptop comes with 32G of ram, the resource-consuming gui and background applications won't allow the benchmark to take all of the memory. Also, the performance will drop significantly if swap kicks in. So I just allocate 24G * 80% for the benchmark and got the N value 45248.

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
45248        Ns
1            # of NBs
224          NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
3            Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

1.3 Performance

1.3.1 Theoretical peak performance

6P: 3.228 GHz * 8 (instructions per cycle, unofficial) * 6 (cores) = 154.944 Gflops

2E: 2.064 GHz * 4 (instructions per cycle, unofficial) * 2 (cores) = 16.512 Gflops

Total: 154.944 Gflops + 33.024 Gflops = 171.456 Gflops

1.3.2 Best performance

```
================================================================================
```

```
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00L2L2       45248   224     2     3             168.26             3.6707e+02
HPL_pdgesv() start time Wed Apr  6 12:36:54 2022

HPL_pdgesv() end time   Wed Apr  6 12:39:42 2022

--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV--VVV-
Max aggregated wall time rfact . . . :               3.65
+ Max aggregated wall time pfact . . :               2.77
+ Max aggregated wall time mxswp . . :               2.55
Max aggregated wall time update  . . :             156.74
+ Max aggregated wall time laswp . . :               5.66
Max aggregated wall time up tr sv  . :               0.16
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   5.61465799e-04 ...... PASSED
```

The result is about 367.07 Gflops, which is actually higher than the theoretical peak performance calculated above. So one possibility is that the theoretical peak performance is incorrect.

If you still want the efficiency: 367.07 / 171.456 * 100% = 214.08%.

2. HPCG

2.1 Environment

Type	Specifications
Compiler	`mpicxx-openmpi-mp` (`Apple clang version 13.1.6 (clang-1316.0.21.2)`)
MPI	Open MPI 4.1.3 from MacPorts

```
mpirun-openmpi-mp -np 6 ./xhpcg
```

2.2 Optimization

The optimization here is limited, as there are not many parameters to adjust. So HPCG used here is compiled directly with `setup/Make.Mac_MPI`.

For `hpcg.dat`, there are only two parameters. One for the local (to an MPI process) dimensions of the problem, and another is the time to run the benchmark. The default dimensions are already reasonable values, and tuning them didn't seem to have much impact on the result.

```
HPCG benchmark input file
Sandia National Laboratories; University of Tennessee, Knoxville
104 104 104
300
```

2.3 Performance

Official results execution time requires at least 1800s, but here I run the benchmark on my laptop and don't have so much time for a sufficient test. Thus the benchmark is run for 300s.

```
Final Summary=
Final Summary::HPCG result is VALID with a GFLOP/s rating of=12.168
Final Summary::HPCG 2.4 rating for historical reasons is=12.2442
```

```
Final Summary::Reference version of ComputeDotProduct used=Performance results are
most likely suboptimal
Final Summary::Reference version of ComputeSPMV used=Performance results are most
likely suboptimal
Final Summary::Reference version of ComputeMG used=Performance results are most
likely suboptimal
Final Summary::Reference version of ComputeWAXPBY used=Performance results are
most likely suboptimal
Final Summary::Results are valid but execution time (sec) is=306.361
Final Summary::Official results execution time (sec) must be at least=1800
```

The final rating is 12.168 GFLOP/s.

3. IO500

3.1 Environment

IO500 git commit: `760bcced8ce8c0d3775910c52c2b77a27fb702df`

3.1.1 Patches to get it to work on macOS

Actually, I don't really understand the `TESTSEXE` part in the Makefile, but it does prevent the compilation.

```
@@ -29,11 +29,11 @@ OBJS = $(patsubst %,./build/%,$(OBJSO))
 TESTS += ini-test
 TESTSEXE = $(patsubst %,./build/%.exe,$(TESTS))

-all: $(VERIFIER) $(PROGRAM) $(TESTSEXE)
+all: $(VERIFIER) $(PROGRAM)

 clean:
    @echo CLEAN
-   @$(RM) ./build/*.o ./build/io500.a *.exe $(PROGRAM)
+   @$(RM) ./build/*.o ./build/io500.a $(PROGRAM)

 ./build/io500.a: $(OBJS)
    @echo AR $@
@@ -56,6 +56,6 @@ $(PROGRAM): ./build/io500.a ./build/main.o
    @echo CC $@
    $(CC) $(CFLAGS) -c -o $@ $<

-./build/%.exe: ./build/%.o $(DEPS) ./build/io500.a
-   @echo LD $@
-   $(CC) -o $@ $< $(LDFLAGS) ./build/io500.a
+# ./build/%.exe: ./build/%.o $(DEPS) ./build/io500.a
+#  @echo LD $@
+#  $(CC) -o $@ $< $(LDFLAGS) ./build/io500.a
```

Other modifications include disabling CUDA and GPUDirect support in ior.

```
diff --git a/prepare.sh b/prepare.sh
index
deeeca40d124c3b686a238f9cededa30c3d9065d..bed4afc3aba5666791a7362c7446ed8bc493d0ba
100755
--- a/prepare.sh
+++ b/prepare.sh
@@ -73,7 +73,7 @@ function build_ior {
   pushd $BUILD/ior
   ./bootstrap
   # Add here extra flags
-  ./configure --prefix=$INSTALL_DIR
```

```
+  ./configure --prefix=$INSTALL_DIR --without-cuda --without-gpuDirect
   cd src
   $MAKE clean
   $MAKE install
```

And a somehow incorrect printf format string (`%ld` to `%llu` for an `uint64_t` field `block_size`).

```
diff --git a/src/phase_ior_rnd4K.c b/src/phase_ior_rnd4K.c
index
6e6b7733e611ca8a0eecfb617a8e529052195a42..a4fc11010ace994065b18eb3b28e918d46723504
100644
--- a/src/phase_ior_rnd4K.c
+++ b/src/phase_ior_rnd4K.c
@@ -54,7 +54,7 @@ void ior_rnd4K_add_params(u_argv_t * argv){
   u_argv_push_printf(argv, "stoneWallingStatusFile=%s/ior-rnd4K.stonewall",
opt.resdir );
   u_argv_push(argv, "-k");
   u_argv_push(argv, "-t=4096");
-  u_argv_push_printf(argv, "-b=%ld", d.block_size);
+  u_argv_push_printf(argv, "-b=%llu", d.block_size);
   u_argv_push_printf(argv, "-s=%d", 10000000);
 }

diff --git a/src/phase_ior_rnd1MB.c b/src/phase_ior_rnd1MB.c
index
956c80389ae934092a8c26e1839990c018be88d8..48e679c31e7860940ac17cbcf0019532e17bfebe
100644
--- a/src/phase_ior_rnd1MB.c
+++ b/src/phase_ior_rnd1MB.c
@@ -54,7 +54,7 @@ void ior_rnd1MB_add_params(u_argv_t * argv){
   u_argv_push_printf(argv, "stoneWallingStatusFile=%s/ior-rnd1MB.stonewall",
opt.resdir );
   u_argv_push(argv, "-k");
   u_argv_push(argv, "-t=1048576");
-  u_argv_push_printf(argv, "-b=%ld", d.block_size);
+  u_argv_push_printf(argv, "-b=%llu", d.block_size);
   u_argv_push_printf(argv, "-s=%d", 10000000);
 }
```

Last some unsupported flags of `clang`.

```
diff --git a/Makefile b/Makefile
index
ee5cee90065a37c63701495a4283e85f0ef0f478..832b97fad98db01f009a2af22634ec77b89bc85e
100644
--- a/Makefile
+++ b/Makefile
@@ -1,8 +1,8 @@
-CC = mpicc
-CFLAGS += -std=gnu99 -Wall -Wempty-body -Werror -Wstrict-prototypes -
Werror=maybe-uninitialized -Warray-bounds
+CC = mpicc-openmpi-mp
+CFLAGS += -std=gnu99 -Wall -Wempty-body -Werror -Wstrict-prototypes -Warray-
bounds

 IORCFLAGS = $(shell grep CFLAGS ./build/ior/src/build.conf | cut -d "=" -f 2-)
-CFLAGS += -g3 -lefence -I./include/ -I./src/ -I./build/pfind/src/ -
I./build/ior/src/
+CFLAGS += -g3 -I./include/ -I./src/ -I./build/pfind/src/ -I./build/ior/src/
 IORLIBS = $(shell grep LDFLAGS ./build/ior/src/build.conf | cut -d "=" -f 2-)
 LDFLAGS += -lm $(IORCFLAGS) $(IORLIBS) # -lgpfs # may need some additional flags
as provided to IOR
```

```
mpiexec-openmpi-mp -np 2 ./io500 config-scc.ini
```

3.2 Optimization

3.2.1 config-scc.ini

After spending about one hour to get the project to be built on macOS, I ran it with the default config and found it trying to write 9920000m of data to my disk. The documentation said "the following variables could be adjusted in order to achieve the minimum 300 seconds execution time", but it is not realistic to write so much data to a laptop, so the value is reduced to 60000m.

```
diff --git a/config-scc.ini b/config-scc.ini
index
9f3acbb15dce89b1c2cec2941d92efd086c364a0..4630508d0448d043ad783f2828d1975bb85abaa6
100644
--- a/config-scc.ini
+++ b/config-scc.ini
@@ -17,7 +17,7 @@ stonewall-time = 30
 [ior-easy]
 API =
 transferSize = 2m
-blockSize = 9920000m
+blockSize = 60000m
 filePerProc = TRUE
 uniqueDir = FALSE
 run = TRUE
```

3.3 Performance

```
IO500 version io500-isc22_v1-26 (standard)
[RESULT]       ior-easy-write        1.571745 GiB/s : time 31.520 seconds
[RESULT]    mdtest-easy-write       12.435453 kIOPS : time 31.378 seconds
[      ]            timestamp        0.000000 kIOPS : time 0.000 seconds
[RESULT]       ior-hard-write        0.123752 GiB/s : time 63.144 seconds
[RESULT]    mdtest-hard-write       10.464419 kIOPS : time 31.153 seconds
[RESULT]                 find      409.563585 kIOPS : time 1.702 seconds
[RESULT]        ior-easy-read        6.163118 GiB/s : time 8.039 seconds
[RESULT]     mdtest-easy-stat       78.305225 kIOPS : time 5.827 seconds
[RESULT]        ior-hard-read        0.596221 GiB/s : time 13.106 seconds
[RESULT]     mdtest-hard-stat       83.661543 kIOPS : time 4.778 seconds
[RESULT]   mdtest-easy-delete       20.308605 kIOPS : time 19.607 seconds
[RESULT]     mdtest-hard-read       29.783319 kIOPS : time 11.600 seconds
[RESULT]   mdtest-hard-delete       15.287931 kIOPS : time 21.646 seconds
[SCORE ] Bandwidth 0.919467 GiB/s : IOPS 36.612305 kiops : TOTAL 5.802050
```

4. Appendix

This proposal is written in Markdown, and the style is adjusted based on VSCode's default.