计算机体系结构Lab2实验报告

PB19051183 吴承泽

修改相应参数配置

相应配置文件`./configs/common/Options.py`各种参数在文件中的位置如下所示：

- **CPU：DerIvO3CPU(the OOO core)**

```
185   def addCommonOptions(parser):
186       # start by adding the base options that do not assume an ISA
187       addNoISAOptions(parser)
188
189       # system options
190       parser.add_argument("--list-cpu-types",
191                           action=ListCpu, nargs=0,
192                           help="List available CPU types")
193       parser.add_argument("--cpu-type", default="DerivO3CPU",
194                           choices=ObjectList.cpu_list.get_names(),
195                           help="type of cpu to run with")
196       parser.add_argument("--list-bp-types",
197                           action=ListBp, nargs=0,
198                           help="List available branch predictor types")
199       parser.add_argument("--list-indirect-bp-types",
200                           action=ListIndirectBP, nargs=0,
201                           help="List available indirect branch predictor types")
202       parser.add_argument("--bp-type", default=None,
203                           choices=ObjectList.bp_list.get_names(),
204                           help="""
205                           type of branch predictor to run with
206                           (if not set, use the default branch predictor of
207                           the selected CPU)""")
```

- **Compiler：gcc -O3 -ffast-math -ftree-vectorize**

```
1  OPT ?= -O3
2
3  CFLAGS   = $(OPT) -ffast-math -ftree-vectorize
4
5  EXEC=lfsr merge mm sieve spmv
6
7  all: $(EXEC)
8
9  %: %.c *.h
10         $(CC) -o $@ $< $(CFLAGS)
11
12 clean:
13         @rm -f $(EXEC)
14
15 gen_arr:
16         python rand_c_arr.py --len=8192 --range=1000000
17         python rand_spmv_arrs.py
```

- **Frequency: 1Ghz**

```
226     parser.add_argument("--l1d-hwp-type", default=None,
227                         choices=ObjectList.hwp_list.get_names(),
228                         help="""
229                         type of hardware prefetcher to use with the L1
230                         data cache.
231                         (if not set, use the default prefetcher of
232                         the selected cache)""")
233     parser.add_argument("--l2-hwp-type", default=None,
234                         choices=ObjectList.hwp_list.get_names(),
235                         help="""
236                         type of hardware prefetcher to use with the L2 cache.
237                         (if not set, use the default prefetcher of
238                         the selected cache)""")
239     parser.add_argument("--checker", action="store_true")
240     parser.add_argument("--cpu-clock", action="store", type=str,
241                         default='1GHz',
242                         help="Clock for blocks running at CPU speed")
243     parser.add_argument("--smt", action="store_true", default=False,
244                         help="""
245                        Only used if multiple programs are specified. If true,
246                        then the number of threads per cpu is same as the
247                        number of programs.""")
```

- **Memory:DDR3_1600_8x8**

```
116     # Memory Options
117     parser.add_argument("--list-mem-types",
118                         action=ListMem, nargs=0,
119                         help="List available memory types")
120     parser.add_argument("--mem-type", default="DDR3_1600_8x8",
121                         choices=ObjectList.mem_list.get_names(),
122                         help="type of memory to use")
123     parser.add_argument("--mem-channels", type=int, default=1,
124                         help="number of memory channels")
125     parser.add_argument("--mem-ranks", type=int, default=None,
126                         help="number of memory ranks per channel")
127     parser.add_argument(
128         "--mem-size", action="store", type=str, default="512MB",
129         help="Specify the physical memory size (single memory)")
130     parser.add_argument("--enable-dram-powerdown", action="store_true",
131                         help="Enable low-power states in DRAMInterface")
132     parser.add_argument("--mem-channels-intlv", type=int, default=0,
133                         help="Memory channels interleave")
```

- **2-level cache hierarchy (64KB L1 icache, 64KB L1 dcache, 2MB L2)**

```
parser.add_argument("--num-l2caches", type=int, default=1)
parser.add_argument("--num-l3caches", type=int, default=1)
parser.add_argument("--l1d_size", type=str, default="64kB")
parser.add_argument("--l1i_size", type=str, default="64kB")
parser.add_argument("--l2_size", type=str, default="2MB")
parser.add_argument("--l3_size", type=str, default="16MB")
parser.add_argument("--l1d_assoc", type=int, default=2)
parser.add_argument("--l1i_assoc", type=int, default=2)
```

- 添加参数设置**IssueWidth**

```python
def addNoISAOptions(parser):
    parser.add_argument("--IssueWidth", type=int, default=8)

    parser.add_argument("-n", "--num-cpus", type=int, default=1)
    parser.add_argument("--sys-voltage", action="store", type=str,
                        default='1.0V',
                        help="""Top-level voltage for blocks running at system
                       power supply""")
    parser.add_argument("--sys-clock", action="store", type=str,
                        default='1GHz',
                        help="""Top-level clock for blocks running at system
                       speed""")
```

```python
for cpu in system.cpu:
    cpu.clk_domain = system.cpu_clk_domain
    #TODO
for cpu in system.cpu:
    cpu.issueWidth = args.IssueWidth
```

修改编译命令：

需测试的配置列表命令如下：

No.	CPU_type	Issue width	CPU_clock	L2 cache
1	O3CPU	8	1GHz	No
2	MinorCPU	8	1GHz	No
3	O3CPU	2	1GHz	No
4	O3CPU	8	4GHz	No
5	O3CPU	8	1GHz	256kB
6	O3CPU	8	1GHz	2MB
7	O3CPU	8	1GHz	16MB

命令参数如下：(以第一个参数集跑merge为例)

```
build/X86/gem5.opt configs/example/se.py --cmd=lab2-benchmark/cs251a-microbench-master/merge --cpu-type=DerivO3CPU --cpu-clock='1GHz' --IssueWidth=8 --caches
```

对五个Benchamark以7种不同的参数集运行，得到若干结果，回答下列问题：

1、应该使用什么指标来比较不同系统配置之间的性能？为什么？

应使用 `simSeconds` 或者是 `simTicks` 用于比较不同配置之间的性能。性能度量可通过响应时间来表示，响应时间越短，性能越好。`simSeconds` 表征了通过该系统模拟运行的时长，`simTicks` 表征运行的时钟周期数，因此适合用于比较不同配置之间的性能。

2、是否有任何基准测试受益于删除 L2 缓存？请说明理由。

选择以下两组作为对比：

No.	CPU_type	Issue width	CPU_clock	L2Cache_Size
1	O3CPU	8	1GHz	0
5	O3CPU	8	1GHz	256KB

可以得到每个benchmark的`simSeconds`数据：

	mm	spmv	lfsr	merge	sieve
1	57218000	56887000	57129000	57129000	56779000
5	83642000	83729000	83643000	83643000	82989000

在此下，5个benchmark的基准测试都受益于删除L2缓存。

以Merge为例，对于dcache，即使是否有L2cache对dcache的命中率没有太大的影响，但有未删除L2cache的平均延迟比删除L2Cache的平均延迟要高接近一倍。

```
system.cpu.dcache.demandHits::cpu.data            8091 # number of demand (read+write) hits (Count)
system.cpu.dcache.demandHits::total               8091 # number of demand (read+write) hits (Count)
system.cpu.dcache.overallHits::cpu.data           8091 # number of overall hits (Count)
system.cpu.dcache.overallHits::total              8091 # number of overall hits (Count)
system.cpu.dcache.demandMisses::cpu.data           808 # number of demand (read+write) misses (Count)
system.cpu.dcache.demandMisses::total              808 # number of demand (read+write) misses (Count)
system.cpu.dcache.overallMisses::cpu.data          808 # number of overall misses (Count)
system.cpu.dcache.overallMisses::total             808 # number of overall misses (Count)
system.cpu.dcache.demandMissLatency::cpu.data     73829000 # number of demand (read+write) miss ticks (Tick)
system.cpu.dcache.demandMissLatency::total        73829000 # number of demand (read+write) miss ticks (Tick)
system.cpu.dcache.overallMissLatency::cpu.data    73829000 # number of overall miss ticks (Tick)
system.cpu.dcache.overallMissLatency::total       73829000 # number of overall miss ticks (Tick)
system.cpu.dcache.demandAccesses::cpu.data         8899 # number of demand (read+write) accesses (Count)
system.cpu.dcache.demandAccesses::total            8899 # number of demand (read+write) accesses (Count)
system.cpu.dcache.overallAccesses::cpu.data        8899 # number of overall (read+write) accesses (Count)
system.cpu.dcache.overallAccesses::total           8899 # number of overall (read+write) accesses (Count)
system.cpu.dcache.demandMissRate::cpu.data     0.090797 # miss rate for demand accesses (Ratio)
system.cpu.dcache.demandMissRate::total        0.090797 # miss rate for demand accesses (Ratio)
system.cpu.dcache.overallMissRate::cpu.data    0.090797 # miss rate for overall accesses (Ratio)
system.cpu.dcache.overallMissRate::total       0.090797 # miss rate for overall accesses (Ratio)
system.cpu.dcache.demandAvgMissLatency::cpu.data 91372.524752 # average overall miss latency ((Cycle/Count))
system.cpu.dcache.demandAvgMissLatency::total 91372.524752 # average overall miss latency ((Cycle/Count))
system.cpu.dcache.overallAvgMissLatency::cpu.data 91372.524752 # average overall miss latency ((Cycle/Count))
system.cpu.dcache.overallAvgMissLatency::total 91372.524752 # average overall miss latency ((Cycle/Count))
```

```
system.cpu.dcache.demandHits::cpu.data            8145 # number of demand (read+write) hits (Count)
system.cpu.dcache.demandHits::total               8145 # number of demand (read+write) hits (Count)
system.cpu.dcache.overallHits::cpu.data           8145 # number of overall hits (Count)
system.cpu.dcache.overallHits::total              8145 # number of overall hits (Count)
system.cpu.dcache.demandMisses::cpu.data           795 # number of demand (read+write) misses (Count)
system.cpu.dcache.demandMisses::total              795 # number of demand (read+write) misses (Count)
system.cpu.dcache.overallMisses::cpu.data          795 # number of overall misses (Count)
system.cpu.dcache.overallMisses::total             795 # number of overall misses (Count)
system.cpu.dcache.demandMissLatency::cpu.data     43865000 # number of demand (read+write) miss ticks (Tick)
system.cpu.dcache.demandMissLatency::total        43865000 # number of demand (read+write) miss ticks (Tick)
system.cpu.dcache.overallMissLatency::cpu.data    43865000 # number of overall miss ticks (Tick)
system.cpu.dcache.overallMissLatency::total       43865000 # number of overall miss ticks (Tick)
system.cpu.dcache.demandAccesses::cpu.data         8940 # number of demand (read+write) accesses (Count)
system.cpu.dcache.demandAccesses::total            8940 # number of demand (read+write) accesses (Count)
system.cpu.dcache.overallAccesses::cpu.data        8940 # number of overall (read+write) accesses (Count)
system.cpu.dcache.overallAccesses::total           8940 # number of overall (read+write) accesses (Count)
system.cpu.dcache.demandMissRate::cpu.data     0.088926 # miss rate for demand accesses (Ratio)
system.cpu.dcache.demandMissRate::total        0.088926 # miss rate for demand accesses (Ratio)
system.cpu.dcache.overallMissRate::cpu.data    0.088926 # miss rate for overall accesses (Ratio)
system.cpu.dcache.overallMissRate::total       0.088926 # miss rate for overall accesses (Ratio)
system.cpu.dcache.demandAvgMissLatency::cpu.data 55176.100629 # average overall miss latency ((Cycle/Count))
system.cpu.dcache.demandAvgMissLatency::total 55176.100629 # average overall miss latency ((Cycle/Count))
system.cpu.dcache.overallAvgMissLatency::cpu.data 55176.100629 # average overall miss latency ((Cycle/Count))
system.cpu.dcache.overallAvgMissLatency::total 55176.100629 # average overall miss latency ((Cycle/Count))
```

除此之外，由于使用L2Cache会使其运行时间增加L2Cache的访存时间，且可以观察到未删除L2Cache组，其命中率较低降低了性能，因此会带来额外的访存时间。

```
system.l2.demandHits::cpu.inst                  11  # number of demand (read+write) hits (Count)
system.l2.demandHits::cpu.data                   4  # number of demand (read+write) hits (Count)
system.l2.demandHits::total                     15  # number of demand (read+write) hits (Count)
system.l2.overallHits::cpu.inst                 11  # number of overall hits (Count)
system.l2.overallHits::cpu.data                  4  # number of overall hits (Count)
system.l2.overallHits::total                    15  # number of overall hits (Count)
system.l2.demandMisses::cpu.inst               714  # number of demand (read+write) misses (Count)
system.l2.demandMisses::cpu.data               446  # number of demand (read+write) misses (Count)
system.l2.demandMisses::total                 1160  # number of demand (read+write) misses (Count)
system.l2.overallMisses::cpu.inst              714  # number of overall misses (Count)
system.l2.overallMisses::cpu.data              446  # number of overall misses (Count)
system.l2.overallMisses::total                1160  # number of overall misses (Count)
system.l2.demandMissLatency::cpu.inst        69307000  # number of demand (read+write) miss ticks (Tick)
system.l2.demandMissLatency::cpu.data        43269000  # number of demand (read+write) miss ticks (Tick)
system.l2.demandMissLatency::total          112576000  # number of demand (read+write) miss ticks (Tick)
system.l2.overallMissLatency::cpu.inst       69307000  # number of overall miss ticks (Tick)
system.l2.overallMissLatency::cpu.data       43269000  # number of overall miss ticks (Tick)
system.l2.overallMissLatency::total         112576000  # number of overall miss ticks (Tick)
system.l2.demandAccesses::cpu.inst               725  # number of demand (read+write) accesses (Count)
system.l2.demandAccesses::cpu.data               450  # number of demand (read+write) accesses (Count)
system.l2.demandAccesses::total                 1175  # number of demand (read+write) accesses (Count)
system.l2.overallAccesses::cpu.inst              725  # number of overall (read+write) accesses (Count)
system.l2.overallAccesses::cpu.data              450  # number of overall (read+write) accesses (Count)
system.l2.overallAccesses::total                1175  # number of overall (read+write) accesses (Count)
system.l2.demandMissRate::cpu.inst          0.984828  # miss rate for demand accesses (Ratio)
system.l2.demandMissRate::cpu.data          0.991111  # miss rate for demand accesses (Ratio)
system.l2.demandMissRate::total             0.987234  # miss rate for demand accesses (Ratio)
system.l2.overallMissRate::cpu.inst         0.984828  # miss rate for overall accesses (Ratio)
system.l2.overallMissRate::cpu.data         0.991111  # miss rate for overall accesses (Ratio)
system.l2.overallMissRate::total            0.987234  # miss rate for overall accesses (Ratio)
system.l2.demandAvgMissLatency::cpu.inst 97068.627451  # average overall miss latency ((Cycle/Count))
system.l2.demandAvgMissLatency::cpu.data 97015.695067  # average overall miss latency ((Cycle/Count))
system.l2.demandAvgMissLatency::total    97048.275862  # average overall miss latency ((Cycle/Count))
system.l2.overallAvgMissLatency::cpu.inst 97068.627451  # average overall miss latency ((Cycle/Count))
system.l2.overallAvgMissLatency::cpu.data 97015.695067  # average overall miss latency ((Cycle/Count))
system.l2.overallAvgMissLatency::total   97048.275862  # average overall miss latency ((Cycle/Count))
```

且有Memory访存的延迟差别很小，因此是否具有L2Cache对Memory访存延迟影响较小。

3、在讨论程序的运行行为时，我们会遇到a) memory regularity，b) control regularity，和 c) memory locality，请谈一谈你对他们的理解。

a) *memory regularity*

即内存访问的规律性，在程序运行的一段时间中，内存的访问往往具有一定的规律性，如对数组等连续的存储空间会线性的遍历访存，而不是随机的进行访问。

```
#如下代码，对内存以固定步长寻址访问
int A[n][n];
for(int i = 0;i < n; i++)
{
    A[i][0] = 1;
}
```

b) *control regularity*

即分支控制的规律性，即对于同一个分支指令来说，大部分的判断下会跳转到同一个位置，而不是随机的进入正确或错误的分支。

```
#如下代码，分支仅有一次错误而跳出，而其余情况均为正确
int A[n][n];
for(int i = 0;i < n; i++)
{
    A[i][0] = 1;
}
```

b) *memory locality*

即内存访问的局部性，在程序运行中会倾向于访问曾经访问过的内存地址附近的内存。

```
#如下代码，每次访问数组元素时，所访问的内存于上次所访问的内存所相邻的位置
int A[n][n];
for(int i = 0;i < n; i++)
{
    A[0][i] = 1;
}
```

4、对于这三个程序属性——a) memory regularity，b) control regularity，和 c) memory locality——从 stats.txt 中举出一个统计指标（或统计指标的组合），通过该指标你可以区分一个workload是否具有上述的某一个属性。 （例如，对于control regularity，它与分支指令的数量成反比。但你一定可以想到一个更好的）。

a）*memory regularity*

统计指标：

- **system.cpu.dcache.overall_hits::total**
- **system.cpu.dcache.overall_misses::total**

*Memory Regularity*与Dcache的命中率成正比，且Dcache的命中率可由上述数据计算出来。

b) *control regularity*

统计指标：

- **system.cpu.branchPred.condPredicted**
- **system.cpu.branchPred.condIncorrect**

*Control Regularity*与预测分支的命中率成正比，且预测分支的命中率可以由上述数据计算出来。

c) *Memory Locality*

统计指标：

- **system.cpu.dcache.overall_hits::total**
- **system.cpu.dcache.overall_misses::total**

*Memory Regularity*与Dcache的命中率成正比，且Dcache的命中率可由上述数据计算出来。

5、对于每一个实验中用到的benchmark，描述它的a) memory regularity, b) control regularity, c) locality；解释该benchmark对哪个微架构参数最敏感（换句话说，你认为"瓶颈"是什么），并使用推理或统计数据来证明其合理性。

a) *mm*

由于*mm*对数组的访存占了访问中的绝大部分，且在循环中访存基本为等步长的访问，如最内层循环`m2[k_row + j + jj]`寻址步长为 *sizeof(float)*，其*Memory Regularity* **较强**；其分支全部为for循环分支，其分支的规律性极强（即基本上是满足分支条件跳转，仅有一次不满足分支条件跳转），因此其*control regularity* **较强**；在*mm.c*中，对所有内存的访问均限制在`float m1[N]; float m2[N]; float prod[N];`与其余定义的*int*型变量，因此其*locality* **较强。**

我认为瓶颈为*Cacheline_size*，*Cacheline_size*的表征Cache中单行所缓存的数据量，而由于mm.c中数组的存取占存取数据的绝大部分，对内存的访问局部性很强，提高*CacheLine_Size*可以有效的提高Cache中的命中率，从而降低*MissLatency*。

以`in_O3CPU_1GHz_nocaches_IssueWidth=8_cacheline_size=64`运行得到：

```
1
2 ---------- Begin Simulation Statistics ----------
3 simSeconds                                 0.000057                       # Number of seconds simulated (Second)
4 simTicks                                   57218000                       # Number of ticks simulated (Tick)
5 finalTick                                  57218000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
6 simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
```

命中率如下：

```
301 system.cpu.dcache.demandHits::cpu.data            8145                       # number of demand (read+write) hits (Count)
302 system.cpu.dcache.demandHits::total               8145                       # number of demand (read+write) hits (Count)
303 system.cpu.dcache.overallHits::cpu.data           8145                       # number of overall hits (Count)
304 system.cpu.dcache.overallHits::total              8145                       # number of overall hits (Count)
305 system.cpu.dcache.demandMisses::cpu.data           788                       # number of demand (read+write) misses (Count)
306 system.cpu.dcache.demandMisses::total              788                       # number of demand (read+write) misses (Count)
307 system.cpu.dcache.overallMisses::cpu.data          788                        # number of overall misses (Count)
308 system.cpu.dcache.overallMisses::total             788                       # number of overall misses (Count)
309 system.cpu.dcache.demandMissLatency::cpu.data     43907000                        # number of demand (read+write) miss ticks (Tick)
310 system.cpu.dcache.demandMissLatency::total       43907000                      # number of demand (read+write) miss ticks (Tick)
311 system.cpu.dcache.overallMissLatency::cpu.data     43907000                         # number of overall miss ticks (Tick)
312 system.cpu.dcache.overallMissLatency::total       43907000                       # number of overall miss ticks (Tick)
```

以`in_O3CPU_1GHz_nocaches_IssueWidth=8_cacheline_size=128`运行得到：

```
2 ---------- Begin Simulation Statistics ----------
3 simSeconds                                  0.000048                       # Number of seconds simulated (Second)
4 simTicks                                    47834000                       # Number of ticks simulated (Tick)
5 finalTick                                   47834000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
6 simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
```

命中率如下：

```
301 system.cpu.dcache.demandHits::cpu.data            8510                       # number of demand (read+write) hits (Count)
302 system.cpu.dcache.demandHits::total               8510                       # number of demand (read+write) hits (Count)
303 system.cpu.dcache.overallHits::cpu.data           8510                       # number of overall hits (Count)
304 system.cpu.dcache.overallHits::total              8510                       # number of overall hits (Count)
305 system.cpu.dcache.demandMisses::cpu.data           546                       # number of demand (read+write) misses (Count)
306 system.cpu.dcache.demandMisses::total              546                       # number of demand (read+write) misses (Count)
307 system.cpu.dcache.overallMisses::cpu.data          546                        # number of overall misses (Count)
308 system.cpu.dcache.overallMisses::total             546                       # number of overall misses (Count)
309 system.cpu.dcache.demandMissLatency::cpu.data     33356000                        # number of demand (read+write) miss ticks (Tick)
310 system.cpu.dcache.demandMissLatency::total       33356000                      # number of demand (read+write) miss ticks (Tick)
311 system.cpu.dcache.overallMissLatency::cpu.data     33356000                         # number of overall miss ticks (Tick)
312 system.cpu.dcache.overallMissLatency::total       33356000                       # number of overall miss ticks (Tick)
```

以`in_O3CPU_1GHz_nocaches_IssueWidth=8_cacheline_size=256`运行得到：

```
2 ---------- Begin Simulation Statistics ----------
3 simSeconds                                  0.000044                       # Number of seconds simulated (Second)
4 simTicks                                    43794000                       # Number of ticks simulated (Tick)
5 finalTick                                   43794000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
6 simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
```

命中率如下：

```
301 system.cpu.dcache.demandHits::cpu.data            8798                       # number of demand (read+write) hits (Count)
302 system.cpu.dcache.demandHits::total               8798                       # number of demand (read+write) hits (Count)
303 system.cpu.dcache.overallHits::cpu.data           8798                       # number of overall hits (Count)
304 system.cpu.dcache.overallHits::total              8798                       # number of overall hits (Count)
305 system.cpu.dcache.demandMisses::cpu.data           393                       # number of demand (read+write) misses (Count)
306 system.cpu.dcache.demandMisses::total              393                       # number of demand (read+write) misses (Count)
307 system.cpu.dcache.overallMisses::cpu.data          393                        # number of overall misses (Count)
308 system.cpu.dcache.overallMisses::total             393                       # number of overall misses (Count)
309 system.cpu.dcache.demandMissLatency::cpu.data     30139000                        # number of demand (read+write) miss ticks (Tick)
310 system.cpu.dcache.demandMissLatency::total       30139000                      # number of demand (read+write) miss ticks (Tick)
311 system.cpu.dcache.overallMissLatency::cpu.data     30139000                         # number of overall miss ticks (Tick)
312 system.cpu.dcache.overallMissLatency::total       30139000                       # number of overall miss ticks (Tick)
```

除此之外，我认为该瓶颈还包括*Dcache_size*，在默认情况下*Dcache_size=64KB*而在该程序中

数组`float m1[N];float m2[N];float prod[N];`每个都占*64KB*内存，访问这三个数组会产生大量的Cache冲突，因此在理论上增大*Dcache_size*，可以有效增加冲突的产生次数，降低总的`MissLatency`。

b) *spmv*

*spmv*程序中使用了四个巨大的`Float`型数组，程序中对`val`、`cols`、与`rowDelim`三个数组寻址的规律性较强，均是等步长寻址，其*Memory Regularity* **较强**；而对数组`vec`的访问地址是基于`Cols`的值，随机性较强，其*Memory Regularity* **较弱**；而在*spmv*程序中使用的分支主要是for循环中的分支，其分支的规律性极强（即基本上是满足分支条件跳转，仅有一次不满足分支条件跳转），因此其*control regularity* **较强**；而该程序中的数据主要来自`spmvArr.h`，均使用数组存储，因此其*locality* **较强**。

我认为瓶颈是*Cacheline_size*，与*mm.c*类似，都含有大量的连续存取的数据，*Cacheline_size*的表征Cache中单行所缓存的数据量，而由于*spmv.c*中数组的存取占存取数据的绝大部分，对内存的访问局部性很强，提高*CacheLine_Size*可以有效的提高Cache中的命中率，从而降低*MissLatency*。

以`in_O3CPU_1GHz_nocaches_IssueWidth=8_cacheline_size=64`运行得到：

```
3 simSeconds                                  0.000057                       # Number of seconds simulated (Second)
4 simTicks                                    56887000                       # Number of ticks simulated (Tick)
5 finalTick                                   56887000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
6 simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
```

命中率如下：

```
301 system.cpu.dcache.demandHits::cpu.data            8130                       # number of demand (read+write) hits (Count)
302 system.cpu.dcache.demandHits::total               8130                       # number of demand (read+write) hits (Count)
303 system.cpu.dcache.overallHits::cpu.data           8130                       # number of overall hits (Count)
304 system.cpu.dcache.overallHits::total              8130                       # number of overall hits (Count)
305 system.cpu.dcache.demandMisses::cpu.data           793                       # number of demand (read+write) misses (Count)
306 system.cpu.dcache.demandMisses::total              793                       # number of demand (read+write) misses (Count)
307 system.cpu.dcache.overallMisses::cpu.data          793                        # number of overall misses (Count)
308 system.cpu.dcache.overallMisses::total             793                       # number of overall misses (Count)
309 system.cpu.dcache.demandMissLatency::cpu.data     46541000                        # number of demand (read+write) miss ticks (Tick)
310 system.cpu.dcache.demandMissLatency::total       46541000                      # number of demand (read+write) miss ticks (Tick)
311 system.cpu.dcache.overallMissLatency::cpu.data    46541000                         # number of overall miss ticks (Tick)
312 system.cpu.dcache.overallMissLatency::total      46541000                       # number of overall miss ticks (Tick)
```

以`in_O3CPU_1GHz_nocaches_IssueWidth=8_cacheline_size=128`运行得到：

```
3 simSeconds                                 0.000048                       # Number of seconds simulated (Second)
4 simTicks                                   47743000                       # Number of ticks simulated (Tick)
5 finalTick                                  47743000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
6 simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
```

命中率如下：

```
301 system.cpu.dcache.demandHits::cpu.data            8496                       # number of demand (read+write) hits (Count)
302 system.cpu.dcache.demandHits::total               8496                       # number of demand (read+write) hits (Count)
303 system.cpu.dcache.overallHits::cpu.data           8496                       # number of overall hits (Count)
304 system.cpu.dcache.overallHits::total              8496                       # number of overall hits (Count)
305 system.cpu.dcache.demandMisses::cpu.data            551                       # number of demand (read+write) misses (Count)
306 system.cpu.dcache.demandMisses::total               551                       # number of demand (read+write) misses (Count)
307 system.cpu.dcache.overallMisses::cpu.data           551                        # number of overall misses (Count)
308 system.cpu.dcache.overallMisses::total              551                       # number of overall misses (Count)
309 system.cpu.dcache.demandMissLatency::cpu.data     36110000                        # number of demand (read+write) miss ticks (Tick)
310 system.cpu.dcache.demandMissLatency::total       36110000                      # number of demand (read+write) miss ticks (Tick)
311 system.cpu.dcache.overallMissLatency::cpu.data    36110000                         # number of overall miss ticks (Tick)
312 system.cpu.dcache.overallMissLatency::total      36110000                       # number of overall miss ticks (Tick)
```

以`in_O3CPU_1GHz_nocaches_IssueWidth=8_cacheline_size=256`运行得到：

```
3 simSeconds                                 0.000044                       # Number of seconds simulated (Second)
4 simTicks                                   43666000                       # Number of ticks simulated (Tick)
5 finalTick                                  43666000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
6 simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
```

命中率如下：

```
301 system.cpu.dcache.demandHits::cpu.data            8738                       # number of demand (read+write) hits (Count)
302 system.cpu.dcache.demandHits::total               8738                       # number of demand (read+write) hits (Count)
303 system.cpu.dcache.overallHits::cpu.data           8738                       # number of overall hits (Count)
304 system.cpu.dcache.overallHits::total              8738                       # number of overall hits (Count)
305 system.cpu.dcache.demandMisses::cpu.data            386                       # number of demand (read+write) misses (Count)
306 system.cpu.dcache.demandMisses::total               386                       # number of demand (read+write) misses (Count)
307 system.cpu.dcache.overallMisses::cpu.data           386                        # number of overall misses (Count)
308 system.cpu.dcache.overallMisses::total              386                       # number of overall misses (Count)
309 system.cpu.dcache.demandMissLatency::cpu.data     29433000                        # number of demand (read+write) miss ticks (Tick)
310 system.cpu.dcache.demandMissLatency::total       29433000                      # number of demand (read+write) miss ticks (Tick)
311 system.cpu.dcache.overallMissLatency::cpu.data    29433000                         # number of overall miss ticks (Tick)
312 system.cpu.dcache.overallMissLatency::total      29433000                       # number of overall miss ticks (Tick)
```

除此之外，我认为瓶颈还包括*L2_Cache*的大小与访问延迟，因为*spmv*中使用的数组较大，增大L2_Cache的大小可以对连续数组可以有效的增加其访问的命中率，降低失效开销。

c) *lfsr*

*lfsr*程序使用的一个结构体数组`arr`，而对结构体数组`arr`的访问是近乎与随机的，因此其*Memory Regularity* **较弱**；在整个程序中，分支指令主要是在`lfsr_loop`函数中的`do...while()`语句，其中判断条件`count`的变化是规律增长的，因此其*control regularity* **较强**；而该程序中的数据主要来自`silly_struct arr[ASIZE];`，使用数组存储但因为整型数`lfsr`的访问较为随机，因此其*locality* **较弱**。

我认为瓶颈是*L1_Dcache*的大小和*L2_Cache*的大小和访问延迟，*L1_Dcache*的增大可以降低冲突的产生次数；整个`silly_struct arr[ASIZE]`占用*2048KB*的内存，普通的L1_Dcache基本上很难达到这个大小，出现冲突时，会直接进入*L2_Cache*进行查找，因此*L2_Cache*越大对内存访存的几率越小，*L2_Cache*访问延迟越低性能越好。

d) *merge*

*merge*程序中使用了一个较大的数组`randArr`，在该程序中对`randArr`中的访问基本为等步长访问，因此*Memory regularity* **较强**；分支语句主要在函数`merge`中，其中`while`循环的*control regularity* **很好**，语句`if (numbers[left] <= numbers[mid])`的分支结果依赖于`numbers`数组，因此随机性较强*control regularity* **较弱**；该程序的数据主要存储在`randArr.h`内的随机数数组与临时变量`temp`数组中，主要访问的是数组中的元素，因此*Memory Locality* **较强**。

我认为瓶颈是*D_Cache*的延迟与分支预测的正确率。*merge*程序中的两个数组使用的总内存为*64KB*，因此*D_Cache*的大小足以容纳这两数组，因此降低访问*D_Cache*的延迟可以有效的提高性能；除此之外，函数*merge*中含有大量的判断语句，且*merge*递归调用自己，其调用频率较高，因此提高分支预测的正确率可有效提高正确率。

e) *sieve*

*sieve*程序中在堆中开辟了一个大小为1000000* *sizeof(char)* 大小的数组，地址空间连续且通过循环`for (int i=p*2; i<=n; i += p)` 以p等步长的访问`notprime`地址空间，因此 *Memory regularity* **较强**；程序中的for循环其分支的规律性极强（即基本上是满足分支条件跳转，仅有一次不满足分支条件跳转），因此其*control regularity* **较强**；在*mm.c*中，对所有内存的访问均限制在`float m1[N]; float m2[N]; float prod[N];`与其余定义的*int*型变量，因此其*locality* **较强**。

我认为*D_Cache*大小和*L2_Cache*的大小应是该程序的瓶颈，因为在*sieve*中创建的 `notprime` 数组在for循环中大量地被访问到，而增大D_Cache和L2_Cache可以有效增大程序在这两个Cache的命中率，可以减少失效率，降低失效总延迟。

6、选择一个benchmark，提出一种你认为对该benchmark非常有效的应用程序增强、ISA 增强和微体系结构增强。

以sieve.c为例：

应用程序增强：

通过改写程序：

```
for(int p = 2; p < n; p += 2) {
    total+=!notprime[p] + !notprime[p+1];
}
//减少一半的Branch指令条数，降低运行时间
```

ISA增强：

通过更改并行体系结构，如MIMD等，可以有效增强该benchmark

```
  for(int p = 2; p < n; p++) {
      total+=!notprime[p];
  }
//不存在数据相关，可以并行化处理
```

微体系结构增强：

使用L2_Cache（或增大L2_Cache的大小），可以有效增加命中率，可以减少失效率，降低失效总延迟。