Jonathan LaFiandra

Experiments

<u>GCC</u>:

<u>EXPERIMENT #1:</u>

DEFAULTS:

```
k0: 3
k1: 2
k2: 1
F: 4
ROB: 12
PREGs: 32
```

Default Values give a IPC of 2.933067 with 34094 total cycles.

MINIMUM(k0+k1+k2+F):

```
k0: 3
k1: 2
k2: 1
F: 4
ROB: 12 = 2*(k0+k1+k2)
PREGs: 32
```

Default Values give a IPC of 2.933067 with 34094 total cycles.

My minimum value was (k0+k1+k2+F) = 10 in the default configuration. Any other configuration does not worth effectively. It seems that GCC is heavily reliant on k0 and fetch, with k1 also being used fairly often as taking away a unit from k1 and giving it to k0 greatly hurt IPC as well. It also seems that k0 = 3 is a fairly sweet spot, as taking away from F or k1 to improve k0 further will always result in a loss of IPC.

<u>EXPERIMENT #2:</u>

DEFAULTS:

```
k0: 3
k1: 2
k2: 1
F: 4
ROB: 12
PREGs: 32
```

Default Values give a IPC of 2.933067 with 34094 total cycles.

MINIMUM(ROB):

```
k0: 3
k1: 2
k2: 1
F: 4
ROB: 12
PREGs: 32
```

These minimum values give a IPC of `2.933067` with `34094` total cycles.

My minimum value was that the ROB size = 12 because I found that lowering the ROB at all will cause a decrease in performance past 98%. Just lowering the ROB to 11 already makes it only 95% effective. This is likely because instructions are not retired super quick; making it seem likely that this code has a great deal of pure dependencies based on the importance of the ROB.

GOBMK:

EXPERIMENT #1:

DEFAULTS:

```
k0: 1
k1: 2
k2: 3
F: 4
ROB: 12
PREGs: 32
```

Default Values give a IPC of `1.600487` with `62481` total cycles.

MINIMUM(k0+k1+k2+F):

```
k0: 2
k1: 2
k2: 1
F: 3
ROB: 10 = (k0+k1+k2)*2;
PREGs: 32
```

These minimum values give a IPC of `1.569243` with `63725` total cycles.

My minimum value was (k0+k1+k2+F) = 8. This seems to be the optimum setup for the trace at just about 98.04% of the default IPC. This distribution shows me that both k0 and k1 are important FUs in this trace, while k2 is once again not nearly as used. It also appears that Fetch is more often than not limited to 3 due to other factors rather than the fetch width itself considering moving it down from 4 to 3 did not greatly affect performance.

EXPERIMENT #2:

DEFAULTS:

```
k0: 1
k1: 2
k2: 3
F: 4
ROB: 12
PREGs: 32
```

Default Values give a IPC of `1.600487` with `62481` total cycles.

MINIMUM(ROB):

```
k0: 3
k1: 2
k2: 1
F: 4
ROB: 8
PREGs: 32
```

These minimum values give a IPC of `1.573787` with `63541` total cycles.

My minimum value was that the ROB size = 8 which is right about at 99.33% as effective as the default values; If you do lower the ROB to 7, it falls right below the cutoff line. This likely means that the major limiting factor is Pregs on the number of instructions to fetch as the ROB isn't really used to its full potential capacity most of the time. Perhaps instructions retire fairly quickly, but Pregs are not freed very often.

Hmmer:

EXPERIMENT #1:

DEFAULTS:

```
k0: 3
k1: 2
k2: 1
F: 4
ROB: 12
PREGs: 32
```

Default Values give a IPC of `2.756263` with `36281` total cycles.

MINIMUM(k0+k1+k2+F):

```
k0: 2
k1: 2
k2: 2
F: 3
ROB: 12 = (k0+k1+k2)*2;
```

PREGs: 32

These minimum values give a IPC of 2.725464 with 36691 total cycles.

My minimum value was (k0+k1+k2+F) = 9. This setup gives a 98.8% effectiveness. It seems that unlike the other traces, k2 is utilized quite a bit in this one. It also seems likely that something besides the fetch width is truly being the limiter on this trace.

EXPERIMENT #2:

DEFAULTS:
k0: 3
k1: 2
k2: 1
F: 4
ROB: 12
PREGs: 32

Default Values give a IPC of 2.756263 with 36281 total cycles.

MINIMUM(ROB):
k0: 1
k1: 2
k2: 3
F: 4
ROB: 11
PREGs: 32

These minimum values give a IPC of 2.720792 with 36754 total cycles.

My minimum value was that the ROB size = 11 which is right about at 98.71% as effective as the default values. With this trace, it seems as though the space in the ROB is extremely important, probably meaning that instructions are not retired super quick. It seems likely that it isn't as reliant on the ROB as gcc, but it still makes sense for there to be a lot of dependencies in order for the ROB to remain so full.

MCF:

EXPERIMENT #1:

DEFAULTS:
k0: 3
k1: 2
k2: 1
F: 4

ROB: 12
PREGs: 32

Default Values give a IPC of `0.590898` with `169234` total cycles.

MINIMUM(k0+k1+k2+F):
k0: 2
k1: 2
k2: 1
F: 3
ROB: 10 = (k0+k1+k2)*2;
PREGs: 32

These minimum values give a IPC of `0.580511` with `172262` total cycles.

My minimum value was (k0+k1+k2+F) = 8. This setup gives a 98.2% effectiveness. MCF is a particularly brutal trace on our machine, falling all the way below 1 PC. Something very obviously heavily limits MCF fetches, and based on how far we can reduce the ROB, it is very likely Pregs are held as occupied for a very long time with this trace.

<u>EXPERIMENT #2:</u>

DEFAULTS:
k0: 3
k1: 2
k2: 1
F: 4
ROB: 12
PREGs: 32

Default Values give a IPC of `0.590898` with `169234` total cycles.

MINIMUM(ROB):
k0: 1
k1: 2
k2: 3
F: 4
ROB: 5
PREGs: 32

These minimum values give a IPC of `0.579710` with `172500` total cycles.

My minimum value was that the ROB size = 5 which is right about at 98.1% as effective as the default values. With this trace, it looks like ROB hardly ever even gets past 4 instructions inside it, meaning instructions are retired very fast, and not fetched particularly fast. This explains why the IPC is so very slow.