Homework 2: Solutions Dominik Gresch

Exercise 2: Polynomials Evaluation

The `poly.c` file can be found in `ex02/poly/`

a) op count for poly:

$$\sum_{i=0}^{N} 1 = N + 1 \qquad \text{adds}$$

$$\sum_{i=0}^{N} \sum_{j=0}^{i-1} 1 = \frac{1}{2}\,(N^2 + N) \qquad \text{mults}$$

$$\Rightarrow \underline{\underline{\frac{1}{2}\,N^2 + \frac{3}{2}\,N + 1 \in \mathcal{O}(N^2)}} \qquad \text{flop}$$

b) Results of the measurement:

size N	128	256	512	1024	2048
performance [flop/cycle]	.197±.003	.198±.007	.148±.004	.111±.002	.128±.002
op count	8385	33153	131841	525825	2100225
runtime [kcycles]	42.6±.6	167±6	890±30	4750±90	16400±300

Table 1: results using `poly`()

System: Intel Core 2 Duo @ 3.06 GHz on Ubuntu 13.10
Compiler: gcc v.4.8.1
Flags: -O3 -fno-tree-vectorize -mno-abm

c) The op count for the `horner`() function is

$$N\,\text{add} + N\,\text{mult} \Rightarrow 2N\,\text{flop}$$

d) Since, in the implementation of the Horner algorithm, the operands for each computation depend on the result of the previous computation, performance will be determined by the latency of those flops.
On my system (Intel Core 2 Duo), the add has a latency of 3 and the mult has a latency of 5. Hence I would expect the runtime to be $8N$ cycles (one add and one mult per loop iteration).

$$\Rightarrow \text{performance} = \frac{2N}{8N} = 0.25\,\text{flop/cycle}$$

e) using the same system, compiler and flags as in a). The results correspond nicely to the value expected in d).

size N	128	256	512	1024	2048
performance [flop/cycle]	.251±.002	.252±.005	.253±.003	.252±.003	.252±.005
op count	256	512	1024	2048	4096
runtime [kcycles]	1.02±.01	2.04±.04	4.05±.05	8.1±.1	16.2±.3

Table 2: results using `horner`()

f) The `horner`() function shows both better performance and better runtime. Whilst the performance is only slightly better ($\sim 1-2\ x$), the lower op count (and especially the better scaling) makes `horner`() perform much better (up to 1000 x for larger N) in terms of runtime.

g) Remark: The $\mathcal{O}(1)$ overhead has not been taken into account for the cost calculation. Measured on the same system as in b).

size N	128	256	512	1024	2048
performance [flop/cycle]	1.38±.03	1.662±.009	1.822±.009	1.895±.009	1.93±.04
op count	256	512	1024	2048	4096
runtime [kcycles]	.186±.004	.308±.002	.562±.003	1.081±.005	2.12±.04

Table 3: results using `horner2`()

The key performance limitation if `horner`() is the sequential dependece of the operands. Hence one can obtain great speedup by using accumulators (up to $\sim 4\ x$ using 8 accumulators).

Exercise 3: Optimization Blockers

The `comp.c` file can be found in `ex03/superslow/`

a) The main performance blockers are:

- calling `f`() twice
- overly complicated `sin`(...)`cos`(...) - expression can be replaced (strength reduction)
- common subexpressions in `f`() (the same `sin`() appears twice)
- in `sqrt`(`fabs`(47 + ..)), `fabs`() is unnecessary because 47 dominates the term (i.e. the right sign can be put manually).

less important:

- scalar replacement
- getting rid of the `if`() (doesn't give much gain due to branch prediction)
- making use of common subexpressions in indices

c) I don't see how one can get rid of the last `sin`(). It might be possible to approximate it well enough via Taylor expansion, but this changes the result, which should not be the goal.

function	perf [Mflops] with −`O0`	perf [Mflops] with −`O3`
`superslow`()	1.89 ± .06	1.87 ± .07
`using_fasterf`()	20.3 ± .5	20.3 ± .5
`fast`()	22.5 ± .1	22.5 ± .1

Table 4: performance of different implementations of the `superslow`() function (averaged)

e) I have made two new versions: one of them only removing what I called the main performance blockers (`using_fasterf`()) and one where I removed all the performance blockers I found (`fast`()). Using all this, I get a speedup of about 12 x. Surprisingly, the difference between `O0` and `O3` is negligible (even in the simpler versions).

Exercise 4: Locality of a Convolution

a) Temporal locality:
For each of the two triple - loops ($\mathtt{i}-\mathtt{j}-\mathtt{k}$), there is the following temporal locality:
Remark: in the ' $\pm$ *', '* $+$ *' stands for the first, '* $-$ *' for the second triple - loop*

- in the write to $\mathtt{B}[\mathtt{i}][\mathtt{j}]$ and the read from $\mathtt{A}[\mathtt{i}\pm 1][\mathtt{j}\pm 2]$ (happens for each $\mathtt{k}$ in the innermost loop).
- if $\mathtt{K}$ is "small enough" (i.e. two iterations of the $\mathtt{j}$ - loop are "soon enough" after each other), there is temporal locality in the read from $\mathtt{H}[\mathtt{K}-\mathtt{k}-1]$ (happens for every $\mathtt{j}$ in the middle loop). This would also occur between the two triple-loops.
- in the $\mathtt{i}$ - and $\mathtt{j}$ - loop, there is temporal locality in $\mathtt{A}$ whenever $\mathtt{i}\pm 1=\mathtt{i}'\pm\mathtt{k}'/\mathtt{K}$ and $\mathtt{j}\pm 2=\mathtt{j}'\pm\mathtt{k}'\%\mathtt{K}$ (or vice versa), for loop - variables $(\mathtt{i},\mathtt{j},\mathtt{k})$ and $(\mathtt{i}',\mathtt{j}',\mathtt{k}')$ that happen "soon enough" after each other. (in particular for $\mathtt{i}=\mathtt{i}',\mathtt{j}=\mathtt{j}'$).

One can safely assume that there is no temporal locality between the two triple - loops for $\mathtt{A}$ and $\mathtt{B}$ (i.e. the same($\mathtt{B}$)/similar($\mathtt{A}$) $(\mathtt{i},\mathtt{j})$-values do not occur "soon enough")

b) Spatial locality:
For each of the two triple - loops ($\mathtt{i}-\mathtt{j}-\mathtt{k}$), there is the following spatial locality:

- accessing neighbouring elements of $\mathtt{H}$ (in the innermost loop, and also across the two triple - loops)
- in the $\mathtt{j}$ - loop, there is write($\mathtt{B}[\mathtt{i}][\mathtt{j}]$) and read $\mathtt{A}[\mathtt{i}\pm 1][\mathtt{j}\pm 2]$ to/from neighbouring elements (same first column, different row).
- within the same $\mathtt{k}/\mathtt{K}$, $\mathtt{A}[\mathtt{i}\pm\mathtt{k}/\mathtt{K}][\mathtt{j}\pm\mathtt{k}\%\mathtt{K}]$ accesses neighbouring elements of $\mathtt{A}$. (both within the $\mathtt{k}$ - and the $\mathtt{j}$ - loop).
- in general, for two sets of loop variables $(\mathtt{i},\mathtt{j},\mathtt{k})$ and $(\mathtt{i}',\mathtt{j}',\mathtt{k}')$ that happen "soon enough" after each other, there is locality in $\mathtt{A}$ if the distance

$$\mathtt{N}\cdot(\mathtt{i}-\mathtt{i}'\pm 1\mp\mathtt{k}'/\mathtt{K})+\mathtt{j}-\mathtt{j}'\pm 2\mp\mathtt{k}'\%\mathtt{K}$$

(or vice versa) is "near enough".