Self-prefetching Data Structures

Anton Lokhmotov, Department of Computing, Imperial College
180 Queen’s Gate, South Kensington Campus, London, SW7 2AZ
Anton.Lokhmotov@cl.cam.ac.uk

Alastair F. Donaldson, Codeplay Software Ltd.
45 York Place, Edinburgh, EH1 3HP
ally@codeplay.com

The scaling of VLSI technology uncovers new challenges that require careful
rethinking of traditional approaches to hardware (and software!) design. In 10–15
years processors are predicted to consist of hundreds to thousands of processing
‘cores’, varying in complexity and specialisation. The Sony/Toshiba/IBM Cell
processor consisting of a Power Processing Element (PPE) and eight Synergistic
Processing Elements (SPEs) is an early example of such heterogeneity, which we
are also likely to see in the forthcoming Intel Larrabee and AMD Fusion proces-
sors.

Scaling to a large number of processing elements (PEs) mandates that at least
some of the PEs operate on their local memories (‘local store’ in the Cell termi-
nology). Orchestrating data movement between distributed memories, however,
is tedious and error-prone (akin to assembly programming). Moreover, when the
working data set of a PE is too large to fit in its local memory, the programmer
has to resort to complex, low-level data streaming techniques such as double-
buffering, to the detriment of portability and maintainability.

We argue that regular data structures stored in main memory, such as arrays,
images and trees together with standard operations, should define additional op-
erations for automatic pre-fetching into local memory. A PE automatically main-
tains a view into such a data structure, so that data is available locally in time for
processing.

We believe that these ‘self-prefetching data structures’, implemented by ‘ex-
pert’ programmers, will be useful to ‘casual’ programmers who wish to program
modern systems efficiently but without being exposed to low-level system details
(similar to the way libraries are used today). Casual programmers will only need
to tune some parameters to the computational workload of their application.
Example  Consider an averaging filter on a two-dimensional $n \times n$ image, where each output pixel gets the average of a square pane of input pixels.

```c
void mean2d(float in[][], float out[][], int n, int k) {
    int x, y, i, j;
    for (x = k/2; x < n-k/2; ++x) {
        for (y = k/2; y < n-k/2; ++y) {
            out[x][y] = 0;
            for (i = -k/2; i < k/2; ++i) {
                for (j = -k/2; j < k/2; ++j) {
                    out[x][y] += in[x+i][y+j];
                } // j
            } // i
            out[x][y] /= (k * k);
        } // y
    } // x
}
```

Suppose that the image is so large that it does not fit entirely into local memory. The programmer can prefetch input pixels into a moving rectangular window, the size of which is smaller than the image size (so the window fits into local memory) but larger than the pane size (so the current pane’s pixels lie within the window).

The pane moves through the image in a regular way (from left to right for each row and from top to bottom through rows), so the programmer can arrange the window to have prefetched input pixels by the time they are needed by the computation on the current pane. Implementing this correctly and efficiently, however, is not trivial. The programmer needs to consider such issues as exploiting reuse of prefetched data, the optimal data transfer rate and size, the window size, alignment constraints etc.

When using a self-prefetching window, the programmer will need to specify its size and the thickness of its ‘boundary layer’: when the current pane starts accessing pixels in this layer (the window might ‘feel’ this by comparing the pixel coordinates to the window boundaries), the window initiates prefetching new pixels and updates the boundaries when prefetching is complete. For example, if the computation for a pixel takes similar time to doing a reasonably sized (i.e. bearing little overhead) data transfer, the boundary layer can be thin, e.g. just 1–2 pixels: the window can start prefetching data just before using up already fetched data. On the other hand, if the computation is fast, the boundary layer should be thicker to allow for prefetching to complete by the time the data is needed.

Interesting research avenues for self-prefetching data structures include designing such structures, trading off performance against memory size, investigating interaction between multiple structures and automatically tuning parameters for best performance.