Course 4: Vector Parallelism

Kristaps Dzonsons

08 December, 2011

Course site: *http://kristaps.bsd.lv/minicourse_12_2011*

Environment

In prior lectures, we focussed on the environment of high-performance computing:

- **UNIX** an operating system popular in non-desktop environments
- **C** a low-level, minimal programming language
- **hardware** purpose-built, high-performance hardware

In this (mini-)lecture, we continue to consider C programming for high-performance hardware.

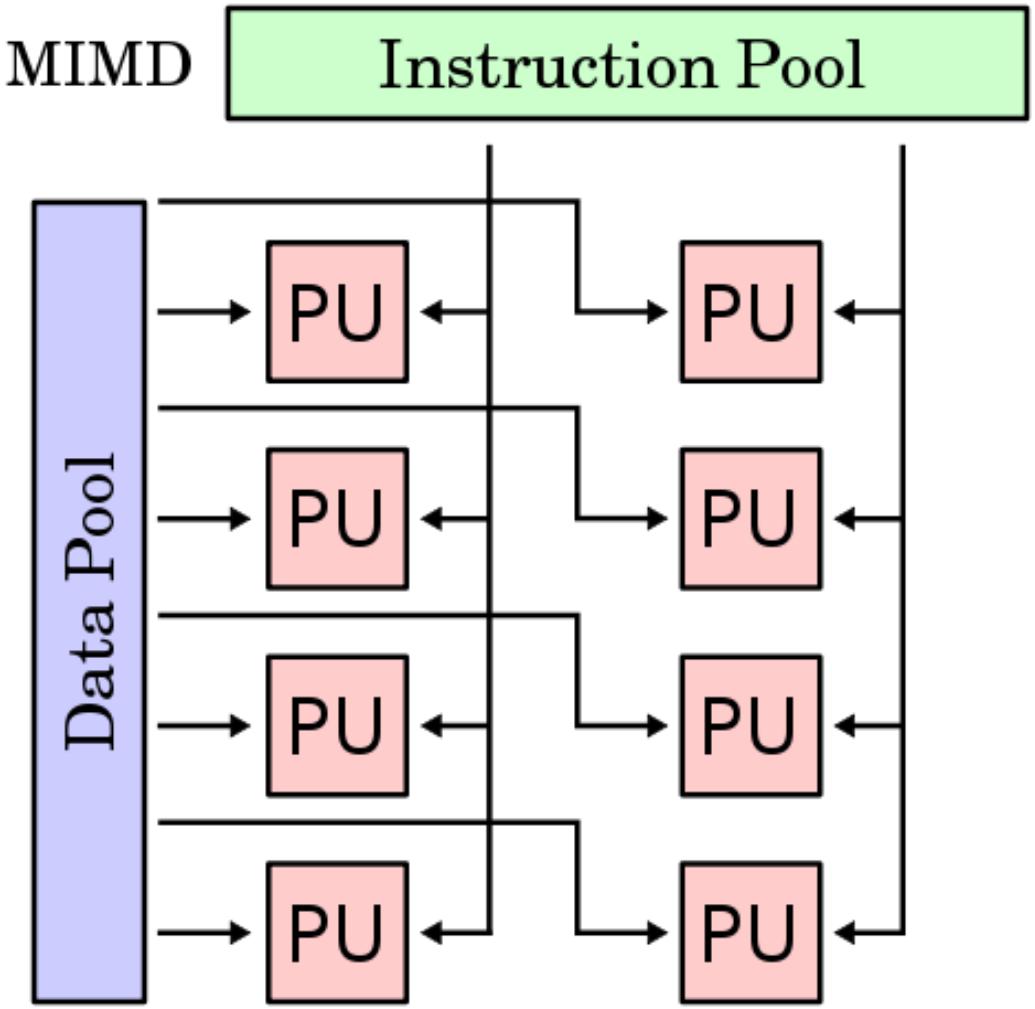
Vector Parallelism: Introduction

Consider again the dot product:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

We've considered both unshared and shared memory high-level parallelism of this equation, which parallelised blocks of multiple instructions (for which we needed multiple cores). Can we do better?

Vector Parallelism: Introduction



Vector Parallelism: Introduction

Our implementation:

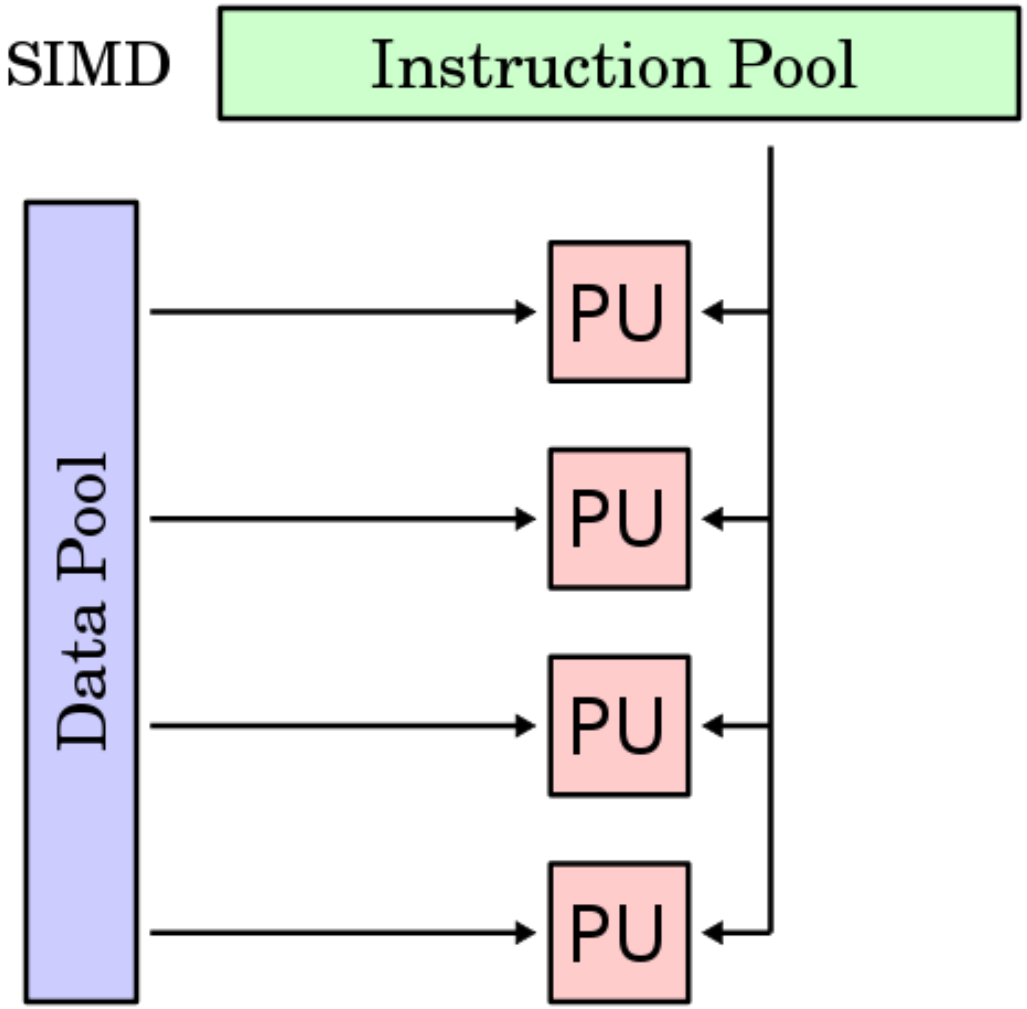
$$\underbrace{a_0 b_0 + \cdots + a_{n/2-1} b_{n/2-1}} + \underbrace{a_{n/2} b_{n/2} + \cdots + a_n b_n}$$

Consider the limit of instruction parallelism:

$$\underbrace{a_0 b_0 + \cdots + a_1 b_1} + \cdots + \underbrace{a_{n-1} b_{n-1} + \cdots + a_n b_n}$$

"Instruction parallelism" is contiguous blocks of identical instructions over different data.

Vector Parallelism: Introduction



Vector Parallelism: Environment

Vector parallelism ("SIMD") takes advantage of chip-level parallelism to execute single instructions in parallel.

There are two main avenues of vector parallelism:

- **FPU** small-scale on-chip parallelism
- **GPU** large-scale off-chip parallelism

In this lecture, we consider FPU parallelism. Both of these have can parallelise a limited set of instructions.

Vector Parallelism: Hardware

Both FPU and GPU vector parallelism require special hardware. Fortunately, FPU parallelism is supported on nearly all modern chips.

AMD
: MMX, 3dNOW!, SSE

Intel
: MMX, SSE

PowerPC
: AltiVec

Vector Parallelism: Software

FPU SIMD instructions can be instrumented in several ways: using a third-party library, direct machine assembly, or compiler extensions.

Consider using compiler extensions to instrument an instruction-parallel dot-product-ish algorithm.

Vector Parallelism: Case Study

```
#include <stdio.h>
#include <stdlib.h>
#ifdef SIMD
#include <xmmintrin.h>
#endif

int
main(void)
{
        float res, *v1, *v2;
#ifdef SIMD
        float buf[4];
        __m128 m1, m2, m3;
        int len;
#endif
        int i, j, sz = 50000000;

        v1 = calloc(sz, sizeof(float));
        v2 = calloc(sz, sizeof(float));

        for (i = 0; i < sz; i++)
                v1[i] = v2[i] = (float)i;
```

Vector Parallelism: Case Study

```
        for (res = 0.0, j = 0; j < 50; j++) {
                i = 0;
#ifdef SIMD
                len = sz - 4;
                for (m3 = _mm_setzero_ps(); i < len; i += 4) {
                        m1 = _mm_load_ps(&v1[i]);
                        m2 = _mm_load_ps(&v2[i]);
                        m3 += m1 * m2;
                }
                _mm_store_ps(buf, m3);
                res += buf[0] + buf[1] + buf[2] + buf[3];
#endif
                for ( ; i < sz; i++)
                        res += v1[i] * v2[i];

        }
        printf("%g\n", res);
        return(EXIT_SUCCESS);
}
```

Vector Parallelism: Case Study

```
% cc -g -W -Wall -o example7 example7.c
% time ./example7
7.55579e+22
    0m23.00s real
    0m22.35s user
    0m0.28s system
% cc -DSIMD -g -W -Wall -o example7 example7.c
% time ./example7
2.08073e+24
    0m11.05s real
    0m10.34s user
    0m0.36s system
```

Vector Parallelism: Case Study

Open questions:

- Can we do better? (In this algorithm, yes.)
- Why are the result values different?