University of Glasgow

Tool Description: Array Programming in Pascal

W. Paul Cockshott, Susanne Oehler, Youssef Gdura, Ciaran McCreesh

Previous Array Pascals

Pascal was one of the first imperative programming languages to be provided with array extensions.

- The first Array Pascal compiler was Actus [Per79, PZA86].
- Turner's Vector Pascal [Tur87], another array extension of the language, was strongly influenced by APL [Ive66].
- Later implementations include
  - Saarbrucken University [KPR92, FOP^{+}92]
  - University of Glasgow [Coc02, CM06]
- Pascal-XSC [HNR92], an extension for scientific data processing, provided extensions for vectors and matrices and interval arithmetic but was not a general array language.

Targets

- Actus targeted distributed memory machines.
- The Turner and Saarbrucken compilers aimed at attached vector accelerators.
- The Glasgow implementation has targeted modern SIMD chips [Coo05, Jac04, Gdu12, CK11] and multi-core chips.

Implicit Parallelism

The Glasgow Vector Pascal compiler uses implicit parallelism:

```
type t = array[1..100, 0..63] of real;
procedure foo(var a, b, c : t);
begin
    a := b * c;
end;
```

to operate on all corresponding elements of the three arrays.

Meaning of Parallelism

This is semantically equivalent to:

```
procedure foo(var a, b, c : t);
var iota: [0..1] of integer;
begin
    for iota[0] := 1 to 100 do
        for iota[1] := 0 to 63 do
            a[iota[0], iota[1]] :=
                b[iota[0], iota[1]] *
                c[iota[0], iota[1]];
end;
```

Iota

The index vector `iota` is implicitly declared with sufficient elements to index the array on the left of the assignment scope, covering the right of the assignment statement.

Note that Perott's # notation is not supported. Instead index sets are usually elided provided that the corresponding positions in the arrays are intended.

`Iota` can be used explicitly to perform things like circular shifts:

```
a := b * c[iota[0], (iota[1]+1) mod 64];
```

Multi-core

Compiling for a 6 core Xeon using AVX transforms the code into:

```
procedure foo(var a, b, c : t);
    procedure stub(start: integer);
    var iota: [0..1] of integer;
    begin
        for iota[0] := start + 1 step 6 to 100 do
            for iota[1] := 0 step 8 to 63 do
                a[iota[0], iota[1] .. iota[1]+7] :=
                    b[iota[0], iota[1] .. iota[1]+7] *
                    c[iota[0], iota[1] .. iota[1]+7];
    end;
var j : integer;
begin
    for j := 0 to 5 do post_job(@stub, %ebp, j);
    for j := 0 to 5 do wait_on_done(j);
end;
```

Reduce

Any binary operator ∘ can be used as a reduction by typing \∘:

```
type r = array[0..63] of real;
function zot(p: real; q: r): real;
begin
    zot := p + \* q
end;
```

`zot` returns the scalar `p` added to the product of the elements of `q`.

Map

```
var a, b, c : array[1..100] of r;
begin
    a := zot(b, c)
end;
```

It is mapped over `b,c` as follows:

```
for iota[0] := 1 to 100 do
    for iota[1] := 0 to 63 do
        a[iota[0],iota[1]] := zot(
            b[iota[0], iota[1]],
            c[iota[0]]);
```

Other Features

- Permutations
- Transpositions
- Bitset operations

Implementation

- The compiler is in Java and is released via SourceForge under the GPL.
- It uses the gcc toolchain for linking.
- It targets a range of contemporary and recent instruction sets: Pentium, Opteron [Jac04], SSE, SSE2, AVX, Playstation 2 (MIPS), Playstation 3 (Cell) [Gdu12], nVidia, and the Intel Knights Ferry [Int14, COX14].
- On Intel AVX and SSE performance is comparable to C with vector intrinsics and threaded building blocks [CGK14].
- For GPUs performance is not as good as CUDA.
- Code tends to be more compact than C or CUDA for the same task

Compliance

ISO standard tests:

Compiler	Number of fails	Success rate
Free Pascal 2.6.2	34	80%
Turbo Pascal 7	26	84.7%
Vector Pascal (Xeon Phi)	4	97.6%
Vector Pascal (Pentium)	0	100%

Demo

```
program bar;
type t = array[1..800, 1..1024] of real;
procedure foo(var a, b, c : t);
begin
    a := b * c + c;
end;

var p, q, r : t; i : integer;
begin
    for i := 1 to 100 do foo(p, q, r)
end.
```

Demo

It performs 2 * 800 * 1024 * 100 = 163 million arithmetic operations. We can compile it for Pentium code and produce a LaTeX listing file:

```
$ vpc bar -L
Glasgow  Pascal Compiler (with vector exensions)
 11                             bar.pas->TeX
 5 generated                                compiled
as --32  --no-warn -g -o p.o p.asm
gcc -g -m32 -o bar p.o   /home/ciaranm/vectorpascal/mmpc/rtl.c
```

Demo

Running it on an AMD A4:

```
$ time ./bar
real    0m1.888s
user    0m1.870s
sys     0m0.008s
```

Demo

We can now compile it using AVX instructions:

```
$ vpc bar -cpuAVX32
```

This vectorises the code so it runs much faster:

```
$ time ./bar
real    0m0.356s
```

Demo

It can be further accelerated by multicore compilation. Note it is not worth using more than 2 cores on this model of CPU as between the 4 cores there are only 2 vector floating point units.

```
$ vpc bar -cpuAVX32 -cores2
$ time ./bar
real    0m0.300s
```

Code Listings

```
listing of file bar.pas
   +---A 'P' at the start of a line indicates the line has been SIMD parallelis
   |+--An 'M' at the start of a line indicates the line has been multi-core par
   ||
   vv
     program bar;
     type t = array[1..800, 1..1024] of real;
     procedure foo(var a, b, c : t);
     begin
   PM    a := b * c + c;
     end;

     var p, q, r : t; i : integer;
     begin
         for i := 1 to 100 do foo(p, q, r)
     end.
```

Code Listings

```
program bar ;
type
    t = array [1..800, 1..1024] of real ;
    procedure foo ( var a , b , c : t );
    begin
        a← b × c + c;
    end ;
var
    Let p, q, r ∈ t;
    Let i ∈ integer;
begin
    for i← 1 to 100 do foo (p, q, r);
end .
```

Another Benchmark (which is Somewhat Unfair)

Next let's compare the performance of Vector Pascal with C when blurring a 1024×1024 pixel colour image. The same separable convolution algorithm is used in both cases:

```
$ vpc blurtime cconv.c
$ ./blurtime
PASCAL        0.03  per run
C             0.442 per run
```

This is because of MMX saturating arithmetic on pixels.

A Fun Example

```
program roman;
const
    rom: array[0..4] of string[1] = ('C','L','X','V','I');
    numb: array[0..4] of integer = (2,1,1,0,3);
var
    s: string;
begin
    s := numb . rom;
    writeln(s);
end.

$ ./roman
CCLXIII
```

Future Work

- Parallel reductions on arbitrary binary functions.
- Front-end for the Haggis programming language, used for teaching in Scottish schools.
- Prototype Vector C front-end, using Matlab or Cilk style array syntax.

http://dcs.gla.ac.uk/~wpc
wpc@dcs.gla.ac.uk

References I

- P Cockshott, Y Gdura, and Paul Keir, *Array languages and the n-body problem*, Concurrency and Computation: Practice and Experience **26** (2014), no. 4, 935–951.
- W.P. Cockshott and A. Koliousis, *The SCC and the SICSA multi-core challenge*, 4th MARC Symposium, December 2011.
- Paul Cockshott and Greg Michaelson, *Orthogonal parallel processing in vector pascal*, Computer Languages, Systems & Structures **32** (2006), no. 1, 2–41.
- Paul Cockshott, *Vector pascal reference manual*, SIGPLAN Not. **37** (2002), no. 6, 59–81.
- Peter Cooper, *Porting the Vector Pascal Compiler to the Playstation 2*, Master's thesis, University of Glasgow Dept of Computing Science, http://www.dcs.gla.ac.uk/~wpc/reports/compilers/compilerindex/PS2.pdf, 2005.
- William Paul Cockshott, Susanne Oehler, and Tian Xu, *Developing a compiler for the xeonphi (tr-2014-341)*, University of Glasgow, 2014.

References II

- A. Formella, A. Obe, WJ Paul, T. Rauber, and D. Schmidt, *The SPARK 2.0 system-a special purpose vector processor with a VectorPASCAL compiler*, System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on, vol. 1, IEEE, 1992, pp. 547–558.
- Youssef Omran Gdura, *A new parallelisation technique for heterogeneous cpus*, Ph.D. thesis, University of Glasgow, 2012.
- R Hammer, M Neaga, and D Ratz, *Pascal xsc*, New Concepts for Scientific Computation and Numerical Data Processing (1992), 15–44.
- Intel Corporation, *Intel xeon phi product family: Product brief*, April 2014.
- K. Iverson, *A programming language*, Wiley, New York, 1966.
- Iain Jackson, *Opteron Support for Vector Pascal*, Final year thesis, Dept Computing Science, University of Glasgow, 2004.
- Christoph W Kessler, Wolfgang J Paul, and Thomas Rauber, *Scheduling vector straight line code on vector processors*, Code Generation Concepts, Tools, Techniques, Springer, 1992, p. 73..91.

References III

R. H. Perrott, *A Language for Array and Vector Processors*, ACM Trans. Program. Lang. Syst. **1** (1979), no. 2, 177–195.

R. H. Perrott and A. Zarea-Aliabadi, *Supercomputer languages*, ACM Comput. Surv. **18** (1986), no. 1, 5–22.

T Turner, *Vector pascal a computer programming language for the array processor*, Ph.D. thesis, PhD thesis, Iowa State University, USA, 1987.