Writing Shared Memory Parallel Programs in Ada

multitasked Newton's method for power series

Jan Verschelde

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
http://www.math.uic.edu/~jan
janv@uic.edu

www.phcpack.org

Ada devroom, FOSDEM 2020, 1 February, Brussels, Belgium

Outline

1. Introduction
   - motivation and problem statement
   - Newton's method to compute power series expansions
   - multithreading on multicore processors

2. Multitasking in Ada
   - launching a crew of workers
   - writing multitasked code
   - evaluation and differentiation at power series
   - a linear block triangular system

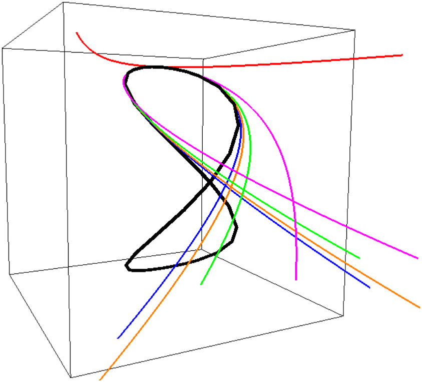
3. Computational Results
   - wall clock times and speedup

motivation: approximation of space curves

Viviani's curve is a space curve defined by

$$\mathbf{f} = (x_1^2 + x_2^2 + x_3^2 - 4, (x_1 - 1)^2 + x_2^2 - 1).$$

At the point $(0, 0, 2)$, consider power series expansions:



Increased degrees of truncation give better approximations.

Newton's method

We compute $\mathbf{x}(t)$ a power series solution to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, starting at a point $\mathbf{x}(0) = \mathbf{z}$, $\mathbf{x}(t) = \mathbf{z} + \mathbf{x}_1 t + \mathbf{x}_2 t^2 + \cdots$.

Let $J_{\mathbf{f}}$ be the matrix of all partial derivatives of $\mathbf{f}$, we compute the update $\Delta\mathbf{x}(t)$ to $\mathbf{x}(t)$ as the solution of a linear system

$$J_{\mathbf{f}}(\mathbf{x}(t))\Delta\mathbf{x}(t) = -\mathbf{f}(\mathbf{x}(t)),$$

and then do $\mathbf{x}(t) := \mathbf{x}(t) + \Delta\mathbf{x}(t)$.

Computational difficulties:

1. increasing number of equations and variables,
2. truncate the power series at increasing degrees,
3. multiprecision arithmetic needed for roundoff errors.

Goal: improve the efficiency by parallel computations.

multithreading on multicore processors

All computers have multicore processors.

Development on three different computers and operating systems:

1. Linux Microway workstation with two 22-core procesors.
   Two 22-core 2.2 GHz Intel Xeon E5-2699, 256 GB RAM.
2. Windows MSI laptop with one 8-core processor.
   Intel Core i9-9880H 2.30 GHz, 32 GB RAM.
3. MacOS X MacBook Pro laptop, with dual core processor.
   Intel Core i7 3.10 GHz processor, 16 GB RAM.

On these three above computers, best speedups are achieved with respectively 88, 16, and 4 threads.

starting worker tasks

procedure `Workers` is instantiated with a `Job` procedure, executing code based on the `id` number.

```
procedure Workers ( n : in natural ) is
   task type Worker ( id,n : natural );
   task body Worker is
   begin
     Job(id,n);
   end Worker;
   procedure Launch_Workers ( i,n : in natural ) is
     w : Worker(i,n);
   begin
     if i < n
      then Launch_Workers(i+1,n);
     end if;
   end Launch_Workers;
begin
   Launch_Workers(1,n);
end Workers;
```

writing multitasked code

We consider memory and granularity when writing multitasked code.

1. memory

   Threads each have a stack, all share the same heap.

   Auxiliary vectors in a computation

   - should not be local variables in a function or procedure,
   - are work space data attributes.

   To avoid race conditions, different tasks work on different data.

2. granularity

   The parallel code defines how jobs are mapped to tasks.

   - Decide on the size of the jobs.
   - A directed acyclic graph defines the order of jobs.
   - Synchronize with relaunching tasks.

3. there are always other issues ...

evaluation and differentiation at power series

For example, evaluate $f = x_1 x_2 x_3 x_4 x_5$,
and compute all its partial derivatives, in three stages:

$$
\begin{array}{lrcl}
\text{1) compute forward products :} & x_1 x_2 & = & x_1 \star x_2 \\
 & x_1 x_2 x_3 & = & x_1 x_2 \star x_3 \\
 & x_1 x_2 x_3 x_4 & = & x_1 x_2 x_3 \star x_4 \\
 & x_1 x_2 x_3 x_4 x_5 & = & x_1 x_2 x_3 x_4 \star x_5 \\
\text{2) compute backward products :} & x_5 x_4 & = & x_5 \star x_4 \\
 & x_5 x_4 x_3 & = & x_5 x_4 \star x_3 \\
 & x_5 x_4 x_3 x_2 & = & x_5 x_4 x_3 \star x_2 \\
\text{3) compute cross products :} & x_1 x_3 x_4 x_5 & = & x_1 \star x_5 x_4 x_3 \\
 & x_1 x_2 x_4 x_5 & = & x_1 x_2 \star x_5 x_4 \\
 & x_1 x_2 x_3 x_5 & = & x_1 x_2 x_3 \star x_5
\end{array}
$$

Every $\star$ is a multiplication of truncated power series.

Every monomial can be evaluated and differentiated independently of every other monomial $\Rightarrow$ straightforward parallelism.

a linear block triangular system

After evaluation and differentiation, we solve $J_{\mathbf{f}}(\mathbf{x}(t))\Delta\mathbf{x}(t) = -\mathbf{f}(\mathbf{x}(t))$.

For example, if we truncate power series at degree 2:

$$\left(A_0 + A_1 t + A_2 t^2\right)\left(\Delta\mathbf{x}_0 + \Delta\mathbf{x}_1 t + \Delta\mathbf{x}_2 t^2\right) = \mathbf{b}_0 + \mathbf{b}_1 t + \mathbf{b}_2 t^2,$$

then in matrix notation, we obtain the block triangular linear system

$$\begin{bmatrix} A_0 & & \\ A_1 & A_0 & \\ A_2 & A_1 & A_0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_0 \\ \Delta\mathbf{x}_1 \\ \Delta\mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}.$$

A coarse grained parallel algorithm applies pipelining.

If the degree of the truncated power series equals d,
using more than d threads will not increase the speedup.

wall clock times and speedup

Wall clock times and speedups are reported on a 10-dimensional system, developing a power series at one cyclic 10-root, a known benchmark.

Running at most 8 steps with Newton's method in quad double precision, for increasing number p of tasks and degree d of truncation:

	$d = 16$		$d = 32$		$d = 64$		$d = 96$	
p	seconds	speedup	seconds	speedup	seconds	speedup	seconds	speedup
1	1.109s		2.304s		20.957s		46.030s	
2	0.649s	1.708	1.376s	1.674	11.582s	1.810	25.491s	1.806
4	0.441s	2.514	0.863s	2.670	7.407s	2.829	16.150s	2.850
8	0.348s	3.186	0.677s	3.405	5.335s	3.928	11.709s	3.931
16	0.376s	2.948	0.727s	3.168	5.279s	3.970	11.684s	3.940

On Windows laptop, Intel Core i9-9880H 2.30 GHz, 8 cores, 32 GB RAM.

The code is available on github at
`https://github.com/janverschelde/PHCpack/`
in the folder `src/Ada/Tasking`.