

Computational Economics: Practical Tools and Techniques Scientific computing (cont.)

Victor V. Zhorin

Computation Institute/BFI

Nov 20, 2013



Key points (review)

- Types of computing: commercial software development vs. scientific research
- From real analysis to numerical methods and computing
- Computer architecture: from naive serial code to vectorization and parallelization
- Low-level tools and techniques: programming languages, math libraries, interactive agile prototyping, high performance computing
- High-level tools and techniques: non-linear and linear optimizers, using domain knowledge for efficient computing
- Parallel programming concepts
- Massively parallel processors \Rightarrow simpler numerical algorithms: larger block-independent data frames with optimal function calls
- All modern CPUs are in a sense massively parallel processors!
- Hybrid computing: single-core HPC, multi-core and cluster computing, many core CPU-GPU and Intel MIC computing

Mapping research ideas to computer architecture

going back and forth between your model and computing implementation

- start mapping your model to computing architecture from the ground up
- single-core → representative optimizing agent with complex dynamics
- multi-core → heterogeneous independent agents processing different shocks under common information set
- fusion systems (clusters with Intel Phi co-processor, CPU-GPU) → hierarchical, multiscale models, micro founded macro models

Math libraries: from real analysis to computing

Implementation matters!

- Intel MKL: industry-standard, can be fine-tuned to use OpenMP, MPI efficiently
- OpenBLAS - multi-threaded high performance implementation for multi-core CPUs
- MacOS X: Accelerate framework
- MATLAB with Intel MKL \Rightarrow 5-6 times faster than open source Fortran/BLAS
- The GNU Scientific Library (GSL) - a C replacement for numerical procedures written in Fortran (Netlib), NO high performance BLAS
- EIGEN templates and Armadillo C++ linear algebra library, the syntax (API) deliberately similar to MATLAB
- R computations with Intel MKL and automatic offloading to Intel Xeon Phi for big data

Scientific computing and business software development

- version control (subversion, git), unit tests, use cases
- modularity, reusability
- need to budget time and resources to comply with the best practices from business software development!
- what went wrong with C++
 - Arrays are not a core part of the language
 - Pointers are everywhere with random holes in memory lanes!
 - Some important features of Fortran-90 only 20 years later added as C/C++ extension in Intel Cilk Plus
- Java: dynamically allocated or resizable arrays \Rightarrow very slow
- CS favorite objects (lists, maps, trees) and concepts (metaprogramming) are huge performance hogs
- Recursions and lambda-calculus \Rightarrow hard to parallelize
- NumPy, MATLAB, FORTRAN - a view over the memory, strided memory model, fast performance lane, no slowing down due to OO design and random pointers

Non-linear Optimization with derivatives:

one solver does not fit all

- Gradients and Hessians are critical for Newton-based NL solvers
- Solution update method: Sequential Quadratic Programming (SQP), Interior-Point (IP)
- Global optimum: trust region, line search
- Penalty function, tolerance, feasibility
- SNOPT
 - line-search SQP; null-space CG option
 - l_1 exact penalty function
- IPOPT - open source in COIN-OR
 - line-search filter algorithm
- KNITRO
 - trust-region Newton, interior with CG option or direct
 - l_1 exact penalty function
 - Active Set - for medium size problems with good initial guess
- If your problem fails to be solved by IPOPT/SNOPT, it might be solved by KNITRO or vice versa

Optimization with weakened assumptions : abandoning convexity requirement!

domain knowledge \Rightarrow efficient numerical algorithms

- MATLAB fminsearch
Nelder-Mead Simplex (slow but reliable)
- Augmented Lagrangian methods (Arrow & Solow, 1958), method of multipliers (Hestens, Powell, 1969) \Rightarrow Pattern Search

$$\underset{X}{\text{maximize}} f(X), \text{ s. t. } h(X) = 0$$

Lagrangian

$$\mathcal{L}(X; \Lambda) = f(X) + \Lambda' h(X)$$

Gradient process

$$\dot{X} = \mathcal{L}_X(X; \Lambda); \dot{\Lambda} = \mathcal{L}_\Lambda(X; \Lambda)$$

Modified (augmented) Lagrangian (based on Lemma by Debreu, Econometrica (1952))

$$\mathcal{L}(X; \Lambda|\theta) = f - \theta h' h + \Lambda' h;$$

Gradient process

$$\dot{X} = f'_X - 2\theta h'_X h + h'_X \Lambda; \dot{\Lambda} = h$$

or set $V = \Lambda + 2\theta \dot{\Lambda}$ then

$$\dot{X} = f'_X + h'_X V$$

Λ represents current market price which rises or falls if excess demand is positive or negative.

V a kind of expected price, based on extrapolation of current rates of change

- HOPSPACK (Hybrid Optimization Parallel Search PACKage)
with asynchronous pattern search solver (supports MPI, OpenMP) over user-defined objective and nonlinear constraint functions (Fortran, C/C++, Perl, MATLAB, Python)

Numerical recipes and domain expertise:

textbook numerical recipes \Rightarrow build up accurate domain guidance

"Redistribution and Social Insurance"

Mikhail Golosov, Maxim Troshkin, Aleh Tsyvinski, 2013.

- finite-horizon discrete-time dynamic programming problem with a three-dimensional continuous state space
- three-stage computational procedure
- shape-preserving least absolute deviation (LAD) value function iteration method with Chebyshev polynomials
- essential to have an efficient and robust optimization algorithm to solve mechanism design problems
- mechanism design problem is a bi-level maximization problem
- the outer-level maximization of the planner has to take into account the best response of the agents, which is the outcome of the inner-level maximization of each agent type with respect to the type reported
- implementation in AMPL/KNITRO
- interior-point optimization with CG iteration for inner-level, active-set with sequential linear quadratic programming iteration at outer level
- globalization strategy - explore multiple feasible starting points
- simple age-dependent linear taxes - welfare loss of 0.9% of consumption equivalent

"Insurance and Taxation over the Life Cycle"

Emmanuel Farhi and Ivan Werning, 2013. Review of Economic Studies, 80.

- It is surprising just how well this relatively simple policy performs. It delivers a welfare gain of 1.47% in lifetime consumption, compared to the 1.56% obtained by the second best. Remarkably, age-dependent linear taxes deliver 95% of the welfare gains of the second-best.
- our characterization of the second best, theoretical and numerical, provides not only useful insights, but can also deliver detailed and surprisingly accurate guidance for simpler tax systems

Non-convex problems: moral hazard and adverse selection

Su-Judd (2007), based on Alexander Karaivanov (2001): "Computing Moral Hazard Programs with Lotteries Using Matlab"

Part 1: Deterministic contract (non-linear program)

- Action-Output

$$\mathcal{A} \otimes \mathcal{Q} : \{a_1, \dots, a_{na}\} \otimes \{q_1, \dots, q_{nq}\}$$

- Stochastic Production Function $p(q|a)$

- Compensation Schedule

$$\mathcal{C}(\mathcal{Q}) = \{c(q_1), \dots, c(q_{nq})\}$$

- Expected utility for the agents $\omega(c, a) = \sum_{q \in \mathcal{Q}} p(q|a)u(c(q), a)$

Deterministic contract (non-linear program)

Principal utility $\mathcal{U}[q - c(a)]$, $q - c(a)$: net profit

$$\begin{aligned} & \underset{(c,a)}{\text{maximize}} && \sum_{q \in \mathcal{Q}} p(q|a) \mathcal{U}[q - c(a)] \\ & \text{s.t.} && \end{aligned}$$

Participation Constraints:

$$\omega(c, a) \geq \omega_0$$

Incentive Compatibility Constraints (ICC):

$$\omega(c, a) \geq \omega(c, \bar{a}), \forall \bar{a} \in \mathcal{A}$$

- Global optimum is not guaranteed
- Sensitive to starting conditions and choice of NL solver

Computing Moral Hazard Programs:

make the problem convex by using lotteries

Part 2: Prescott-Townsend Lotteries (linear program)

- Global optimum (conditional on the grid) is reliably achieved

$$\underset{\pi(q,c,a)}{\text{maximize}} \left[\sum_{Q,C,A} \pi(q,c,a) \mathcal{U}[q-c] \right]$$

- $\pi(q,c,a)$ is a probability distribution

- Participation Constraints:

$$\sum_{Q,C,A} \pi(q,c,a) u(c,a) \geq u_0$$

- Mother Nature/Technology Constraints:

$$\forall \{\bar{q}, \bar{a}\} \in Q \times A$$

$$\sum_c \pi(\bar{q}, c, \bar{a}) = P(\bar{q}|\bar{a}) \sum_{Q,C} \pi(q,c,\bar{a})$$

- Incentive Compatibility Constraints (ICC) for action variables:

$$\forall a, \hat{a} \in A \times A$$

$$\sum_{Q,C} \pi(q,c,a) u(c,a) \geq \sum_{Q,C} \pi(q,c,a) \frac{P(q,\hat{a})}{P(q,a)} u(c,\hat{a})$$

Linear Programming: the choice of solver matters

- MATLAB linprog
- Open source: GLPK, Ip_solve, CLP, SoPlex
- IBM CPLEX , XPRESS
- Gurobi: Interfaces to R, MATLAB, Python
- reliable information is hard to find, obsolescence is an issue
- need to be aware before you know you need it

	running time	instances solved	solved (%)
CBC	10.20	41	47.13
CPLEX	1.45	73	83.91
GLPK	22.11	3	3.45
GUROBI	1.00	77	88.51
LP_SOLVE	19.40	5	5.75
SCIP-C	3.76	63	72.41
SCIP-L	6.40	52	59.77
SCIP-S	5.33	57	65.52
XPRESS	1.29	74	85.06

SCIP-L (using CLP);SCIP-S (using SoPlex)

"Analysis of commercial and free and open source solvers for linear optimization problems", B. Meindl and M.

Hybrid approach to Moral Hazard and Adverse Selection Programs

Su-Judd (2007)

- Step 1: Solve LP in lotteries on coarse grids which guarantees solution that can serve as a good starting point - close to global optimum
- Step 2: Use this information to exclude bad (nonsensical) local traps from non-linear constrained optimization
- Step 3: (locally convergent only!) Combine with multi-start option in non-linear solver to converge quickly on (hopefully) true global optimum
- Step 4: (optional) Iterate
- Step 5: (optional) Do Structural Estimation in parallel

Parallel Programming:

same task done faster or more complex task done in feasible time

SAXPY, single-precision real Alpha X Plus Y (BLAS, level 1):

$$Y \leftarrow \alpha * X + Y$$

where $X_i, Y_i, i \in [1, n]$ - vectors

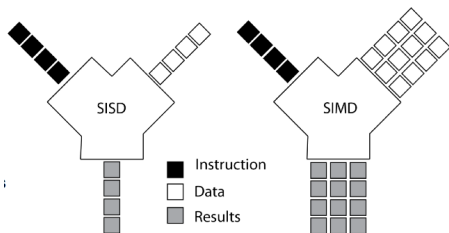
- Instruction (control) parallelism, strong scaling
 - Scalar uniprocessor - $2n$ steps
 - Two functional units (an adder and a multiplier) - $n + 1$ steps, speedup $\frac{2n}{n+1} \approx 2$
 - Amdahl's law:
If s is a fraction of code that is executed serially then speedup from parallelizing $p = 1 - s$ fraction using N processors:

$$Speedup \leq \frac{1}{s + p/N}$$

- Data parallelism: two steps with n processors handling $\alpha * X$ and Y simultaneously, speedup is proportional to $N < n$
- Gustafson-Barsis law, weak scaling:

$$Speedup = s + N * (1 - s)$$

Parallel Programming: types of parallel computing models



- Data parallel - the same instructions are carried out simultaneously on multiple data items (SIMD)
- Task parallel - different instructions on different data (MIMD)
- MIMD: Message passing (MPI) - overlapping computation and communication (!) , MATLAB Distributed Computing Server with Parallel Computing Toolbox
- SIMD: Array Programming (implicit parallelization), NumPy, High Performance Fortran, Vectorization (and **Tensorization**) in Matlab
- Task/data parallel paradigms : OpenMP, Fortran 2008 DO CONCURRENT
- Hybrid Programming: CPU-GPU, Intel Phi MIC architecture, SIMD→OpenMP→MPI

Parallel Programming: MPI in 5 minutes, task parallelism

Dynamic programming: value function iterations, heterogeneous types mapped to multiple processors

```
1 use mpi
2 integer :: nproc, id, ierr, sndr ! MPI
3 integer, dimension(MPI_STATUS_SIZE) :: STATUS ! MPI
4 ! initializing MPI
5 call MPI_INIT(ierr)
6 call MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
7 call MPI_COMM_RANK(MPI_COMM_WORLD, id, ierr)
8
9 do i=n,1,-1 ! state space
10     <compute V(i,t,id+1)>
11 enddo ! state space
12 call MPI_BARRIER(MPI_COMM_WORLD, ierr)
13 do i=1, nproc-1
14     if (id .eq. i) then
15         call MPI_SEND(V, n, MPI_DOUBLE_PRECISION, 0, id, MPI_COMM_WORLD, ierr)
16     end if ! id > 0
17 ...
18     if (id .eq. 0) then
19         call MPI_RECV(V, n, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE, MPI_ANY_TAG,
20             MPI_COMM_WORLD, STATUS, ierr)
21         sndr=STATUS(MPI_SOURCE)
22         V(1:n,t, sndr+1)=V
23     end if !
24     call MPI_BARRIER(MPI_COMM_WORLD, ierr)
25 enddo ! i
26 call MPI_FINALIZE(ierr)
```


Tensors and Tensorization: implicit parallelization

- What is a tensor?
 - Tensor is an element of tensor space
 - Tensor space is a new vector space \mathcal{W} constructed from components of vector spaces, for example, given \mathcal{V}_1 and \mathcal{V}_2 : order two tensor $\mathcal{W} = \mathcal{V}_1 \otimes \mathcal{V}_2$
- Is a tensor a kind of vector? - Yes
- Is a matrix a special kind of tensor? - Yes and No
- With tensorization technique, multidimensional (multivariate) computations (linear programming, dynamic programming, MLE, likelihood ratio statistics) are much faster and more transparent than the corresponding single-dimension (univariate) computations

Hypercubes and hyperspheres:

tensorization and non-cartesian (adaptive, sparse) grids

- tensorization: robust, simple, stable, highly efficient, easily parallelizable brute-force attack
- human desire for more photo and video drives engineers to manufacture more efficient processors
- SIMD: common in modern processors in order to improve the performance of multimedia use (large number of vectors, data frames)
- GPUs and MICs are coming

Computing Moral Hazard Programs: SIMD and Tensorization

- Discretization: C , \mathcal{Q} and \mathcal{A} are finite **ordered** sets.
- Key idea: build up multidimensional tensor object from low-dimensional vector objects while keeping the tensor structure in one-dimensional vector projection, then apply math operations to vectorized tensors
- Tensor product $C \otimes \mathcal{Q} \otimes \mathcal{A}$
- MATLAB Kronecker tensor product: $KRON(X, Y) = X \otimes Y$

```
1 grc = linspace(0,4,41); %consumption
2 grq = [1 4]; %output
3 gra = [0 .2 .4 .6 .8 1]; %action
4 nc = length(grc); nq = length(grq); na = length(gra);
5
6
7 %dimension of lottery vector per type
8 N = na*nq*nc;
9
10 C = kron(ones(1,na*nq),grc);
11 Q = kron(kron(ones(1,na),grq),ones(1,nc));
12 A = kron(gra,ones(1,nc*nq));
13
14 %participation constraints
15 b_neq = -U_0;
16 A_neq = -u(C,A);
17
18 %objective function
19 Obj = Q-C;
```

Parallel Linear Algebra: ScaLAPACK and PETSc

ScaLAPACK:

- extends the LAPACK library to MIMD with distributed memory
- Language : Fortran, interfaces: C, C++, Fortran
- Dense systems
- Support in Commercial Packages: MKL - Intel, IMSL

PETSc:

- Portable Extensible Toolkit for Scientific Computation
- Scalable (parallel) solution of linear and non-linear PDEs
- Sparse systems
- Uses MPI for all parallel communications
- Distributed arrays
- Parallel Krylov subspace methods
- Parallel preconditioners
- Parallel (Newton-based) nonlinear solvers

GPU: OpenACC vs CUDA and OpenCL

- CUDA and OpenCL - highly complex C/Fortran instructions
- OpenACC - directive based standard that provides hints to compiler for a section of code to be offloaded from a host CPU to an attached accelerator.
- OpenMP (fully independent threads) → OpenACC (data dependent)

```
1  subroutine saxpy(n, a, x, y)
2  real(8) :: x(:), y(:), a
3  integer :: n, i
4
5  !OpenMP directive
6  !$omp parallel do
7  !OpenACC directive
8  !$acc kernels
9  do i=1,n
10 y(i) = a*x(i)+y(i)
11 enddo
12 !$acc end kernels
13 !$ omp end parallel do
14 end subroutine saxpy
15 ...
16 $ main program
17 $ call SAXPY on 1M elements
18 call saxpy(2**20, 2.0, data-x, data-y)
```

MAGMA:

- "the number of cores will continue to escalate because of the desire to pack more and more components on a chip while avoiding the power wall, instruction level parallelism wall, and the memory wall"
- "there seems to be no doubt that future generations of computer systems, ranging from laptops to supercomputers, will consist of a composition of heterogeneous components"

M. Baboulin, J. Dongarra, J. Herrmann, and S. Tomov. "Accelerating linear system solutions using randomization techniques." *ACM Transactions on Mathematical Software (TOMS)* 39, no. 2 (2013)

New hybrid/fusion algorithms:

- Iterative MC (not to be confused with Monte Carlo simulations or integrations), main idea - construct **artificial random process** and to prove that the mathematical expectation of the process is equal to the unknown solution (or its functional) of the problem: "Monte Carlo Methods For Applied Scientists" by Ivan T. Dimov, 2005.
- K. Judd, L. Maliar and S. Maliar, (2012). "Merging Simulation and Projection Approaches to Solve High-Dimensional Problems".

Factors of computing performance:

from serial optimization to vectorization and parallelization

- Vectorization 7x by taking advantage of SIMD registers and SIMD instruction, strong scaling
- Parallelization on 16-cores, OpenMP multithreading 19x, weak scaling
- Phi co-processor, 244 threads, OpenMP multithreading 3.3x, weak scaling

Ninja gap: from pricing 4.7 Million options per second to pricing 12.3 Billion options per second

Shuo Li , "Achieving Superior Performance on Black-Scholes Valuation Computing using Intel Xeon Phi Coprocessors", 2013

RCC at University of Chicago (Midway cluster)

ready to access high-performance computing for you

- 284 Shared Compute Nodes, 4544 Cores
- Each node has two eight-core 2.6GHz Intel Xeon E5-2670 "Sandy Bridge" processors with 32GB of main memory
- GPU Computing (GPU), 2 Tesla K20 devices per node
- MIC nodes, 2 Intel Phi devices per node
- Shared-Memory (SM), with 1TB main memory
- R, Python, MATLAB, STATA, IPOPT, Armadillo, Intel MKL, Intel MPI, Intel C++ and Fortran compilers, Portland C++ and Fortran

```
1 #!/bin/bash
2
3 #SBATCH --job-name=test_job
4 #SBATCH --output=test.out
5 #SBATCH --error=test.err
6 #SBATCH --nodes=1-1 --cpus-per-task=12
7 #SBATCH --time=1-12:00:00
8 module load matlab intelmpi
9 matlab -nodisplay -r "test"
10
11 sbatch test.batch
12 salloc --exclusive -n1 srun -n1 -N1 --pty --preserve -env $SHELL
13 scontrol show node midway-g19-01
14 NodeName=midway-g19-01 Arch=x86_64 CoresPerSocket=8
15 CPUAlloc=16 CPUErr=0 CPUTot=16 CPULoad=15.97 Features=lc,e5-2670,32G,noib
16 OS=Linux RealMemory=32000 AllocMem=32000 Sockets=2 Boards=1
```


Large-scale high performance computing resources

- DOE Oak Ridge Titan and TACC Stampede, open to researchers through the U.S. DOE INCITE program and NSF XSEDE program
- Number 2 and number 6 on `top500.org` list of the world's top supercomputers
- 560K cores, 710 terabytes of RAM, 8,209 kW (Titan); 462K cores, 192 terabytes of RAM, 4,510 kW (Stampede)
- Titan: 16C AMD Opteron CPUs, 2.2GHz and NVIDIA Tesla K20 GPU; Stampede : PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Intel Xeon Phi
- 27 Peta(10^{15})FLOPS (Titan), 8.5 PetaFLOPS (Stampede) (your PC \approx 5-20 Giga(10^9)FLOPS)
- bitcoin network: \approx 51 exaFLOPS (Nov 12, 2013)