Parallelizing a Black-Scholes Solver based on Finite Elements and Sparse Grids

PDCoF @ IPDPS 2010

Hans-Joachim Bungartz, Alexander Heinecke, *Dirk Pflüger*, and Stefanie Schraufstetter

Scientific Computing in Computer Science,
Technische Universität München

April 19–23, 2010

Overview

Financial Option Pricing

Options (contracts) reserve the right (no obligation)

- to buy (call option) or sell (put option)
- a certain good (asset, underlying) S
- at some point of time t
- for an agreed price K

Useful, e.g., to limit potential loss (hedge against risks)

Financial Option Pricing

Options (contracts) reserve the right (no obligation)

- to buy (call option) or sell (put option)
- a certain good (asset, underlying) S
- at some point of time t
- for an agreed price K

Useful, e.g., to limit potential loss (hedge against risks)

Many different types. Consider, e.g., expiration time t:

- European options: $t = T$
- American options: $t \in [0, T]$
- Bermudan options: $t \in \{t_0, t_1, \ldots, t_n\}$

(Payoff function at expiration time serves as end condition for pricing)

Financial Option Pricing (2)

Problem

- How to price an option (determine its current fair value $V(\vec{S}, t_0)$)?

Frequently used mathematical model: Black-Scholes equation

- Model underlying stock's price $S(t)$ as stochastic Wiener process

$$\mathrm{d}S(t) = \mu S(t)\mathrm{d}t + \sigma S(t)\mathrm{d}W(t)$$

- Obtain general Black-Scholes equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sum_{i,j=1}^{d} \sigma_i \sigma_j \rho_{ij} S_i S_j \frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{i=1}^{d} \mu_i S_i \frac{\partial V}{\partial S_i} - rV = 0$$

with volatilities σ_i, drifts μ_i, risk-free interest rate r, d stocks S_i

Determining the Option Price

In general, no closed form solution

- Price stochastically (MC techniques)
  - Easy to use, implement, parallelize
  - Scaling independent of dimensionality
  - Low(er) convergence rates
  - Greeks (derivatibes) costly to compute

Determining the Option Price

In general, no closed form solution

- Price stochastically (MC techniques)
  - Easy to use, implement, parallelize
  - Scaling independent of dimensionality
  - Low(er) convergence rates
  - Greeks (derivatibes) costly to compute
- Price numerically (discretize PDE via finite differences/ elements/volumes)
  - Hard to derive and solve PDE formulation for complex options
  - Suffer curse of dimensionality
  - Fast convergence rates
  - Greeks faster to derive

Numerical Solution with Finite Elements

- Employ spatial FE discretization
  - Restrict solution to finite dimensional subspace V_N,

$$V(\vec{S}, t) := \sum_{i=1}^{N} \alpha_i(t)\varphi_i(\vec{S}) \in V_N$$

  - Obtain time-dependent system of linear equations

$$B\frac{\partial}{\partial \tau}\vec{\alpha}(\tau) = -\frac{1}{2}\sum_{i,j=1}^{d}\sigma_i\sigma_j\rho_{ij}C\vec{\alpha} + \sum_{i=1}^{d}\left(\mu_i - \frac{1}{2}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{ij}(1+\delta_{ij})\right)D\vec{\alpha} + rB\vec{\alpha}$$

with, e.g., $B_{p,q} := \langle\varphi_p, \varphi_q\rangle_{L^2}$

- Discretize time (Euler/Crank-Nicolson/...)
  - Solve PDE backward in time $\tau := T = t, \quad t = t_0, t_1, \ldots, T$

Sparse Grids (1)

Problem: curse of dimensionality

- Straightforward spatial discretization with $h = n^{-1}$ fails: $\mathcal{O}(n^d)$ grid points

Therefore: sparse grids

- Reduce $\mathcal{O}(n^d)$ to $\mathcal{O}(n \log(n)^{d-1})$
- Similar accuracy

Sparse Grids (1)

Problem: curse of dimensionality

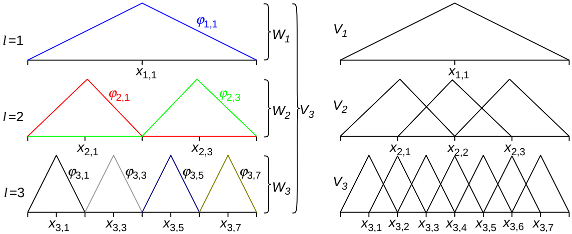
- Straightforward spatial discretization with $h = n^{-1}$ fails: $\mathcal{O}(n^d)$ grid points

Therefore: sparse grids

- Reduce $\mathcal{O}(n^d)$ to $\mathcal{O}(n \log(n)^{d-1})$
- Similar accuracy

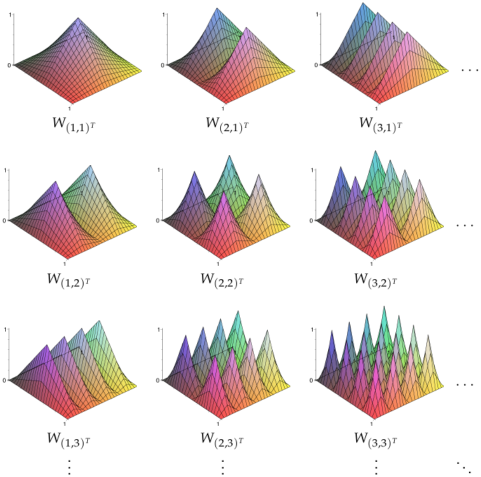
Basic idea:

1) Hierarchical basis in 1d (here: piecewise linear)



Sparse Grids (2)

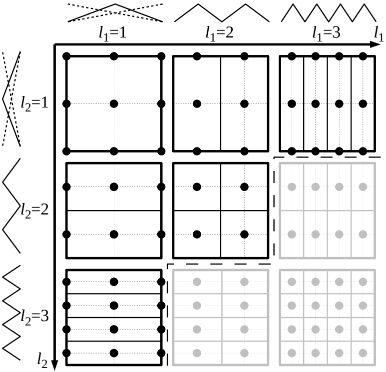
2) Extension to *d*-dimensional basis functions via tensor product approach

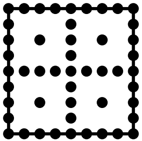


Sparse Grids (3)

Sparse grid space $V_n^{(1)}$ (take only most important sub spaces):

$$V_n^{(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}$$



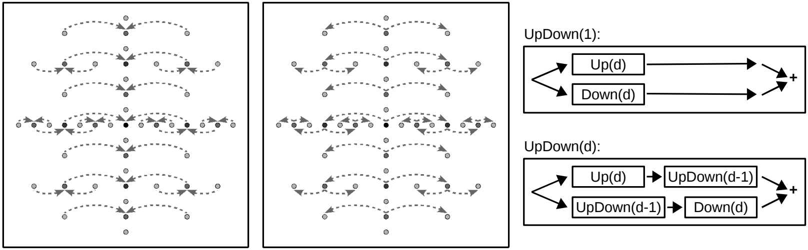


$V_3^{(1)}$

Parallelization

Parallelization on shared memory systems (multi-/many-core)

- Difficult to parallelize (no data/domain splitting)
- Application of matrices requires multi-recursive algorithms



- Parallelization of critical parts using OpenMP 3.0's task concept
  - New task for each recursive descend

Parallel Results

Hardware

- Mobile Intel Penryn Core2Duo (2×2.26 GHz)
- Two-socket Intel Nehalem (8×2.93 GHz, Quick Path Interconnect)
- Two-socket AMD Shanghai (8×2.4 GHz, Hypertransport)
- Two-socket AMD Istanbul (24×2.6 GHz, Hypertransport)

- Multi-socket systems all NUMA
- Measure parallel efficiency on n cores

$$E_n := \frac{t_1}{t_n \cdot n}$$

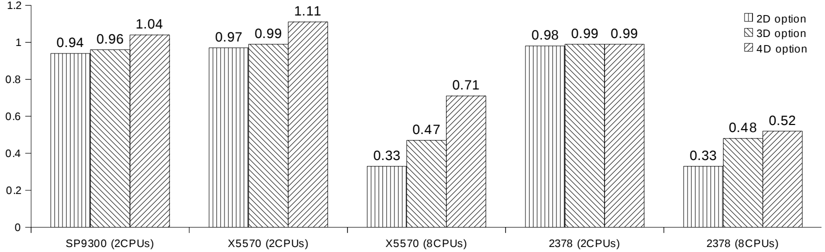
Parallel Results (2)

Example: Intel Xeon X5570 (Nehalem)

option type		1 thread	2 threads		4 threads		8 threads	
d	r	t_1 (s)	t_2 (s)	E_2	t_4 (s)	E_4	t_8 (s)	E_8
2	0.00	580	300	0.97	220	0.66	220	0.33
	0.05	610	310	0.98	230	0.66	230	0.33
3	0.00	3,060	1,540	0.99	950	0.81	810	0.47
	0.05	3,060	1,540	0.99	970	0.79	810	0.47
4	0.00	26,860	12,100	**1.11**	6,960	0.97	4,760	0.71
	0.05	26,900	12,150	**1.11**	7,000	0.96	4,790	0.70
5	0.00	176,700					23,600	0.94

- Task size has to be big enough
- Super-linear speed-up possible (cache sharing)

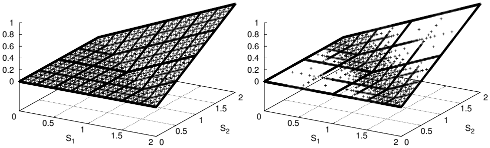
Parallel Results (3)



- Parallelization strongly memory bounded
- Memory access equally distributed
- Intel's 32 KB 8-way level-one cache better suited than AMD's 64 KB 2-way level-one cache
- Similarly QPI better suited than Hypertransport

Conclusions and Future Work

- Sparse Grids enable FE discretizations in dimensions $d > 3$
- Parallelization of multi-recursive algorithms with OMP tasks
- Strongly memory bounded
  - Parallel efficiency depends on chache-associativity
  - and bandwith of memory access
- First experiments:
  - Adaptively refined sparse grids
  - OMP's built-in task load balancing works very well



Conclusions and Future Work

- Sparse Grids enable FE discretizations in dimensions $d > 3$
- Parallelization of multi-recursive algorithms with OMP tasks
- Strongly memory bounded
  - Parallel efficiency depends on chache-associativity
  - and bandwith of memory access
- First experiments:
  - Adaptively refined sparse grids
  - OMP's built-in task load balancing works very well

Ongoing and future work:

- Even higher-dimensional options (spatial adaptivity, optimize algorithms)
- Improve memory access pattern