Fast and smooth simulation of space-time problems

Day 3'

Department of Computer Science
AGH University of Science and Technology, Kraków, Poland
home.agh.edu.pl/paszynsk

Outline

- Isogeometric finite element method
- Alternating Directions Implicit (ADI) method
- Isogeometric L2 projections
- Explicit dynamics
- Example 1: Heat transfer
- Installation of IGA-ADS solver
- Parallel shared memory explicit dynamics
- Example 2: Non-linear flow in heterogenous media
- **Parallel distributed memory explicit dynamics**
- Example 3: Linear elasticity
- Implicit dynamics
- Example 4: Implicit heat transfer
- Example 5: Pollution problem
- Labs with implict dynamics

Parallel domain decomposition based explicit dynamics

Parallel version for distributed-memory machines Message Passing Interface (**MPI**)

Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Calo, *Parallel Fast Isogeometric Solvers for Explicit Dynamics*, Computing and Informatics 36(2) (2017)

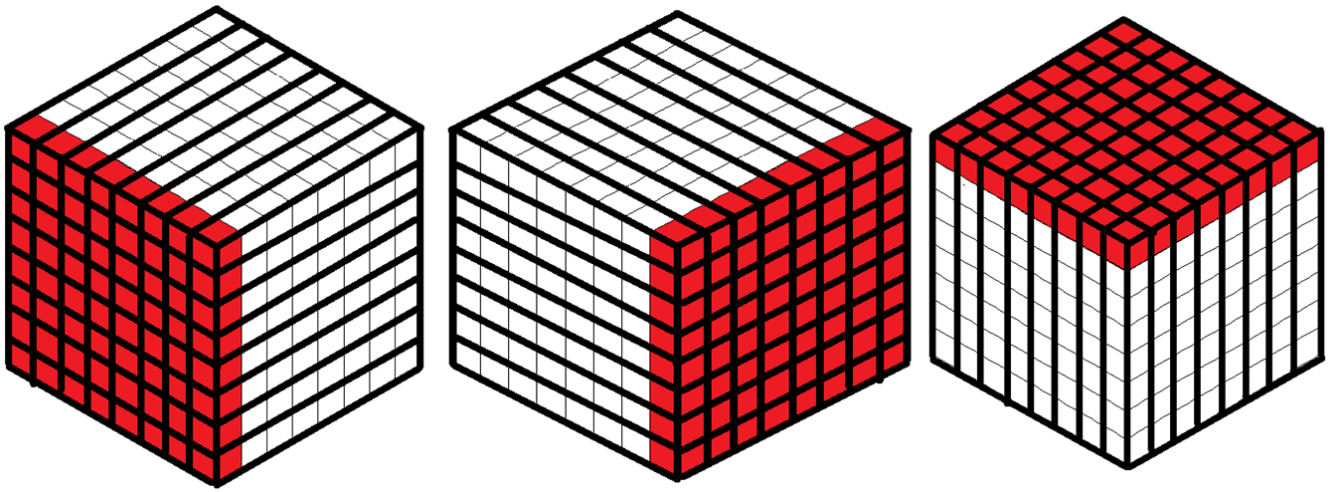


Figure: Cube of processors. Gathers and scathers RHSs in three faces to perform three 1D solves with multiple RHS.

Gather data

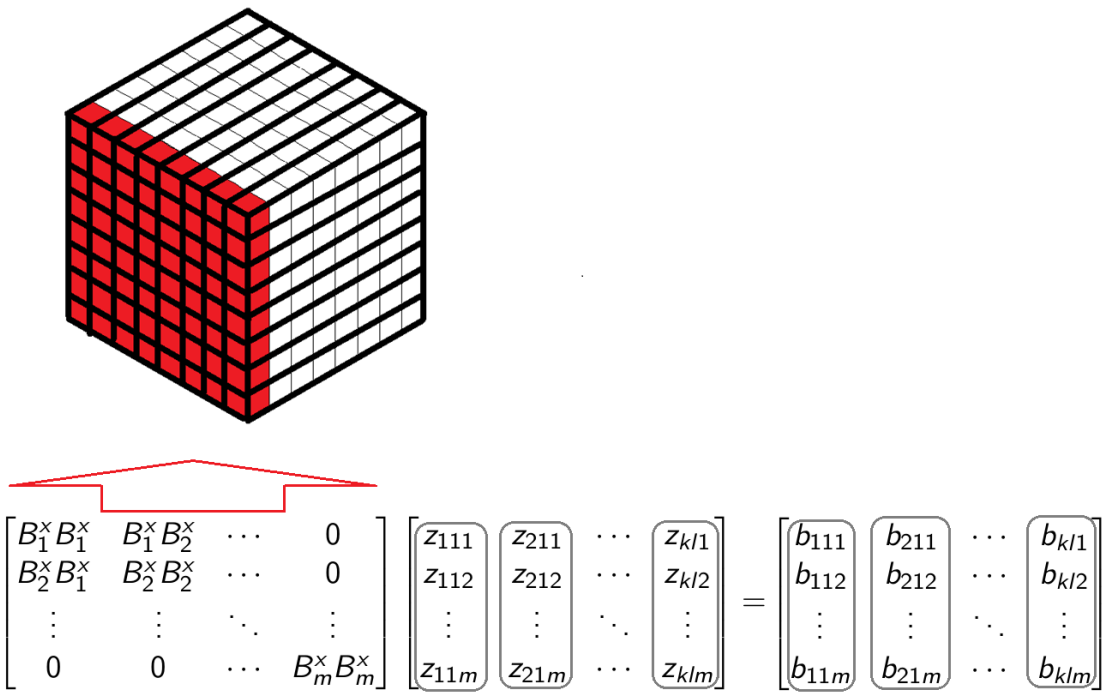
Message Passing Interface (**MPI**)

```
subroutine Gather
  (F, F_out, n, elems, stride, dims, shifts, comm, ierr)

call Linearize(F_out_lin,F_out,n+1,stride)

call mpi_gatherv (F_lin,          ←data to send by each processor 1, .., N_x
  elems * stride,                 ←size of data assigned to this processor
  MPI_DOUBLE_PRECISION,           ←type of data
  F_out_lin,                      ←data to receive by processor 0 in a row
  dims,                           ←size of data from entire row
  shifts,                         ←offsets of slices from all processor from a row
  MPI_DOUBLE_PRECISION,           ←type of data to receive
  0, COMM, ierr)                  ←communicator along a row

call Delinearize(F_out_lin,F_out,n+1,stride)
```

Parallel domain decomposition based explicit dynamics

Parallel version for distributed-memory machines (Fortran, MPI)

Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Calo, *Parallel Fast Isogeometric Solvers for Explicit Dynamics*, Computing and Informatics 36(2) (2017)



$$\begin{bmatrix} B_1^x B_1^x & B_1^x B_2^x & \cdots & 0 \\ B_2^x B_1^x & B_2^x B_2^x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_m^x B_m^x \end{bmatrix} \begin{bmatrix} z_{111} & z_{211} & \cdots & z_{kl1} \\ z_{112} & z_{212} & \cdots & z_{kl2} \\ \vdots & \vdots & \ddots & \vdots \\ z_{11m} & z_{21m} & \cdots & z_{klm} \end{bmatrix} = \begin{bmatrix} b_{111} & b_{211} & \cdots & b_{kl1} \\ b_{112} & b_{212} & \cdots & b_{kl2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{11m} & b_{21m} & \cdots & b_{klm} \end{bmatrix}$$

Figure: Gather and scatter data into three faces on cube of processors

Parallel domain decomposition based explicit dynamics

Parallel version for distributed-memory machines (Fortran, MPI)
Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Calo, *Parallel Fast Isogeometric Solvers for Explicit Dynamics*, Computing and Informatics 36(2) (2017)

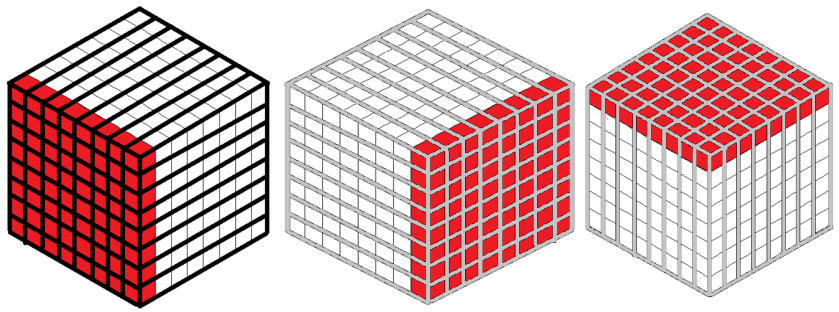


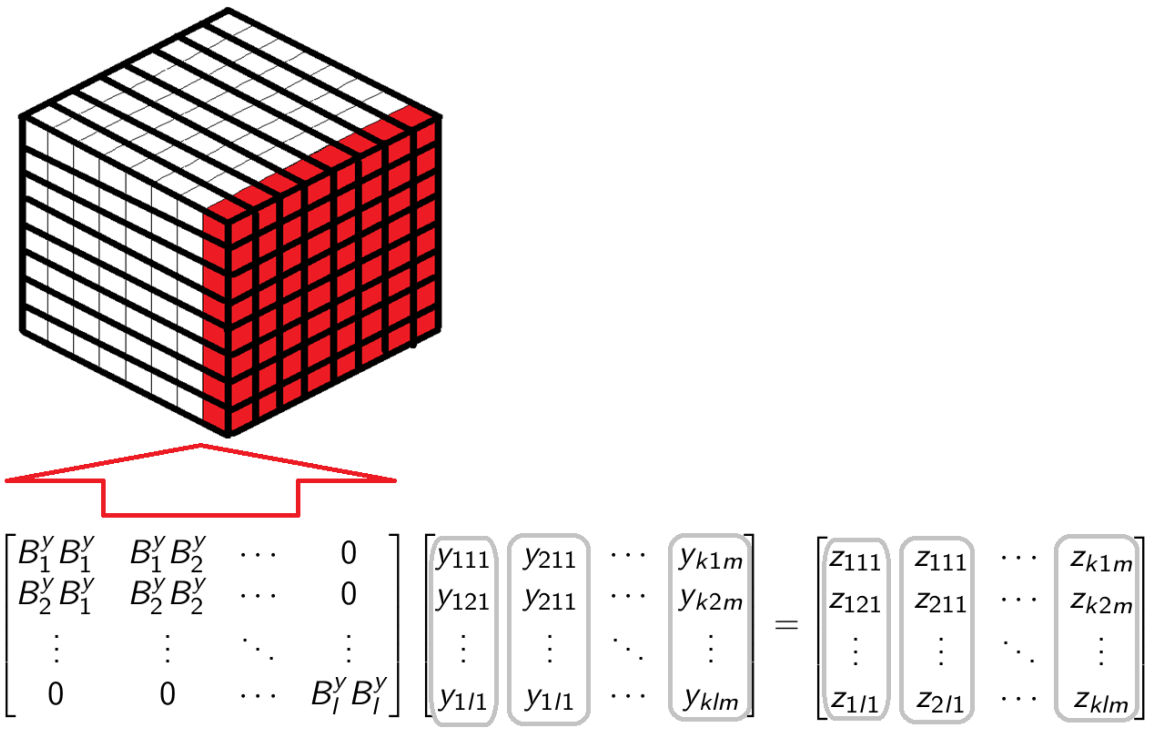
Figure: Gather into OYZ face

0. Integration
1a. Gather in every row of processors into OYZ face
1b. Solve $N_y N_z$ $1D$ problems with multiple RHS
1c. Scatter results onto cube of processors
1d. Reorder right hand sides

Parallel domain decomposition based explicit dynamics

Parallel version for distributed-memory machines (Fortran, MPI)

Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Calo, *Parallel Fast Isogeometric Solvers for Explicit Dynamics*, Computing and Informatics 36(2) (2017)



$$\begin{bmatrix} B_1^y B_1^y & B_1^y B_2^y & \cdots & 0 \\ B_2^y B_1^y & B_2^y B_2^y & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_l^y B_l^y \end{bmatrix} \begin{bmatrix} y_{111} & y_{211} & \cdots & y_{k1m} \\ y_{121} & y_{211} & \cdots & y_{k2m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1l1} & y_{1l1} & \cdots & y_{klm} \end{bmatrix} = \begin{bmatrix} z_{111} & z_{111} & \cdots & z_{k1m} \\ z_{121} & z_{211} & \cdots & z_{k2m} \\ \vdots & \vdots & \ddots & \vdots \\ z_{1l1} & z_{2l1} & \cdots & z_{klm} \end{bmatrix}$$

Figure: Gather into *OXZ* face

Parallel domain decomposition based explicit dynamics

Parallel version for distributed-memory machines (Fortran, MPI)
Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Calo, *Parallel Fast Isogeometric Solvers for Explicit Dynamics*, Computing and Informatics 36(2) (2017)

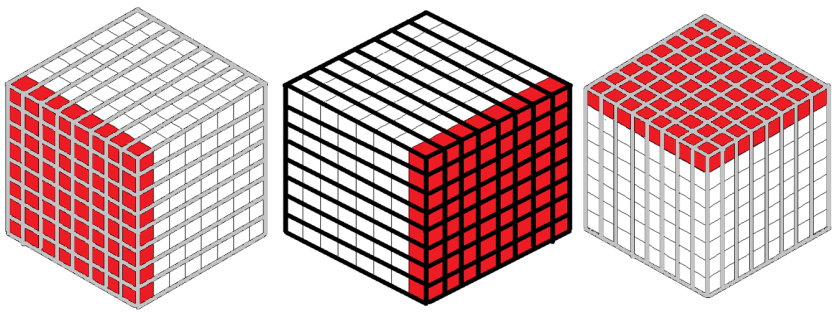


Figure: Gather and scatter data into three faces on cube of processors

2a. Gather in every row of processors into OXZ face
2b. Solve $N_x N_z$ $1D$ problem with multiple RHS
2c. Scatter results onto cube of processors
2d. Reorder right hand sides

Parallel domain decomposition based explicit dynamics

Parallel version for distributed-memory machines (Fortran, MPI)

Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Calo, *Parallel Fast Isogeometric Solvers for Explicit Dynamics*, Computing and Informatics 36(2) (2017)

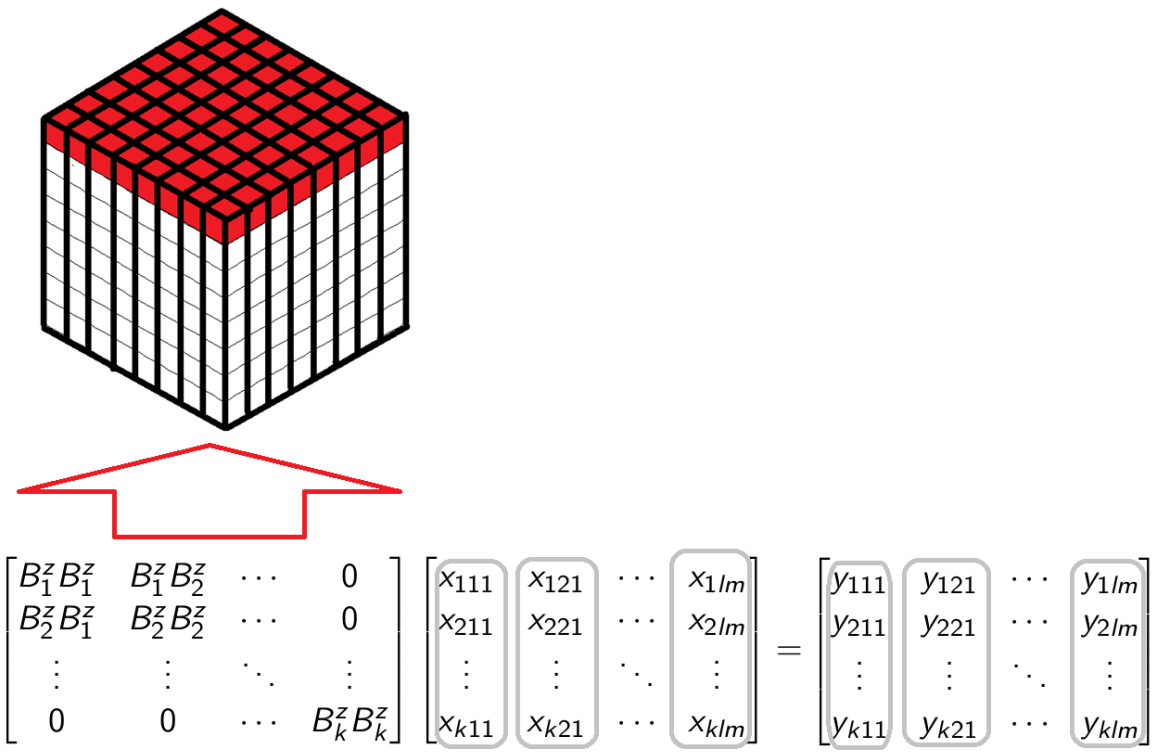


Figure: Gather into *OXY* face

Parallel domain decomposition based explicit dynamics

Parallel version for distributed-memory machines (Fortran, MPI)
Maciej Woźniak, Marcin Łoś, Maciej Paszyński, Lisandro Dalcin, Victor Calo, *Parallel Fast Isogeometric Solvers for Explicit Dynamics*, Computing and Informatics 36(2) (2017)

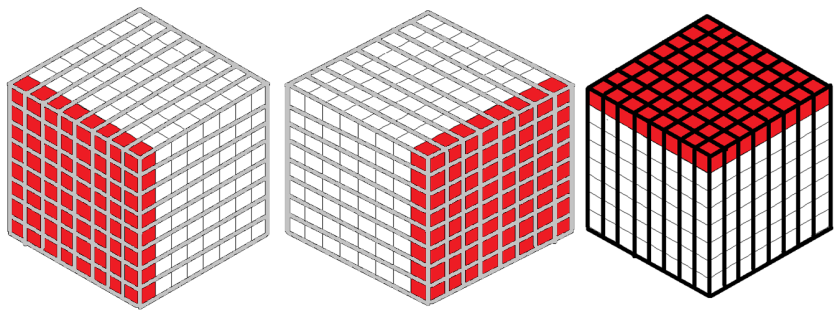


Figure: Gather and scatter data into three faces on cube of processors

3a. Gather in every row of processors into OXY face
3b. Solve $N_x N_y$ $1D$ problem with multiple RHS
3c. Scatter results onto cube of processors
3d. Reorder right hand sides

Parallel domain decomposition based explicit dynamics

We have a mesh of $N_x \times N_y \times N_z$ elements

There are $(p_x+1)(p_y+1)(p_z+1)$ basis functions over each element

Integration over all elements $O(\frac{p_x^2 p_y^2 p_z^2 N_x N_y N_z}{c_x c_y c_z})$

Solution (1D problem multiple RHS) $O\left(\frac{(p_x^2 c_x + p_y^2 c_y + p_z^2 c_z)(N_x N_y N_z)}{c_x c_y c_z}\right)$

Gather $O\left(\frac{(c_x+c_y+c_z) N_x N_y N_z}{c_x c_y c_z}\right)$

Reorder $O\left(\frac{N_x N_y N_z p_x p_y p_z}{c_x c_y c_z}\right)$

Scather $O\left(\frac{(c_x+c_y+c_z) N_x N_y N_z}{c_x c_y c_z}\right)$

Assuming

$N_x = N_y = N_z = N^{1/3},\ p_x = p_y = p_z = p\ c_x = c_y = c_z = c^{1/3}$

we have the following cost $\left(\frac{p^6 N}{c} + \frac{p^2 N}{c^{2/3}} + \frac{p^3 N}{c}\right) t_{comp} + \left(\frac{N}{c^{2/3}}\right) t_{comm}$

which implies **the computational complexity** $O\left(p^6 \frac{N}{c}\right)$

and **the communication complexity** $O\left(\frac{N}{c^{2/3}}\right)$

Numerical experiments LONESTAR Linux cluster

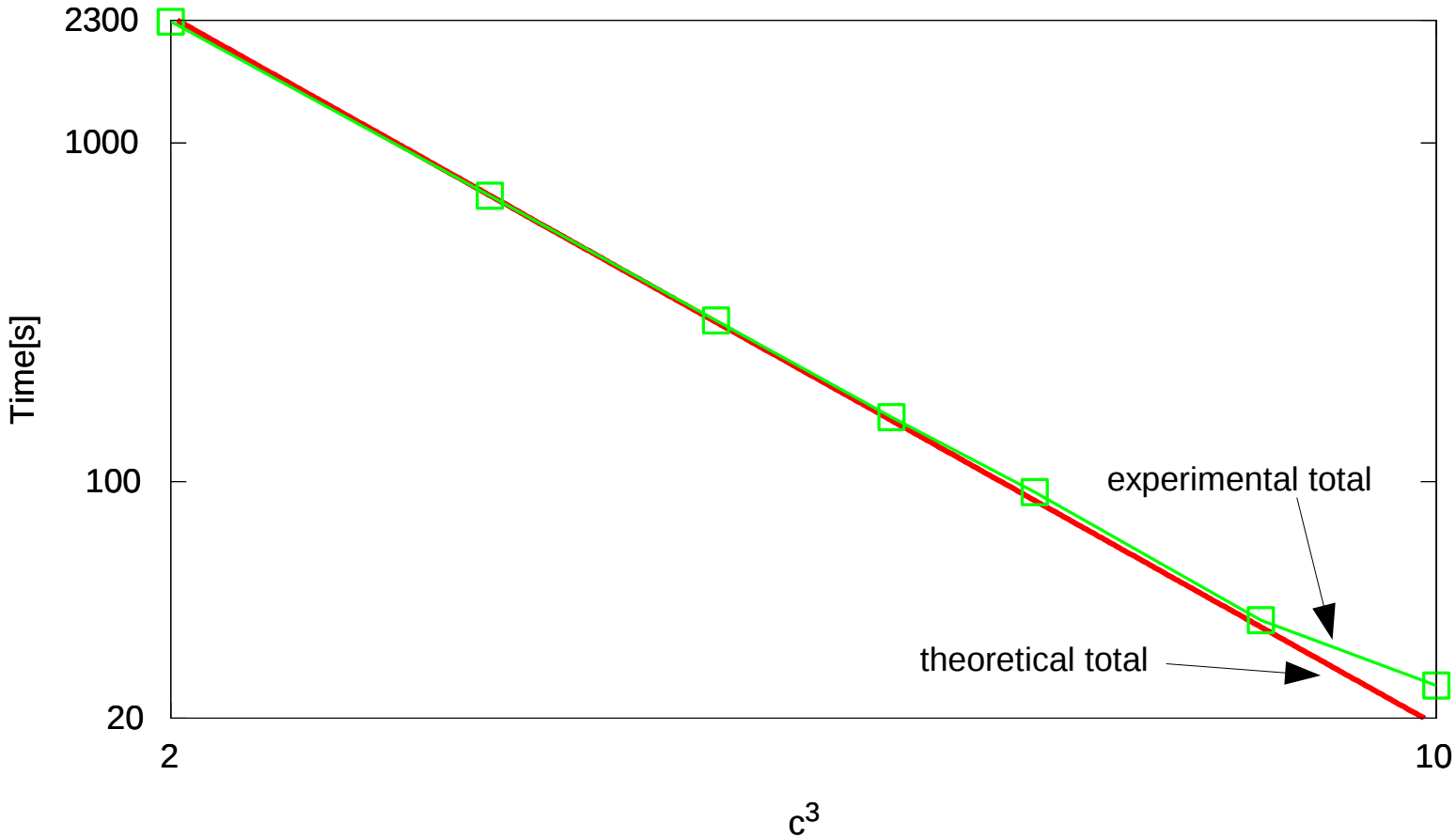


Figure: Comparison of total experimental and theoretical execution time for $N = 512$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Parallel domain decomposition based explicit dynamics

The integration time is dominating the solution time significantly.

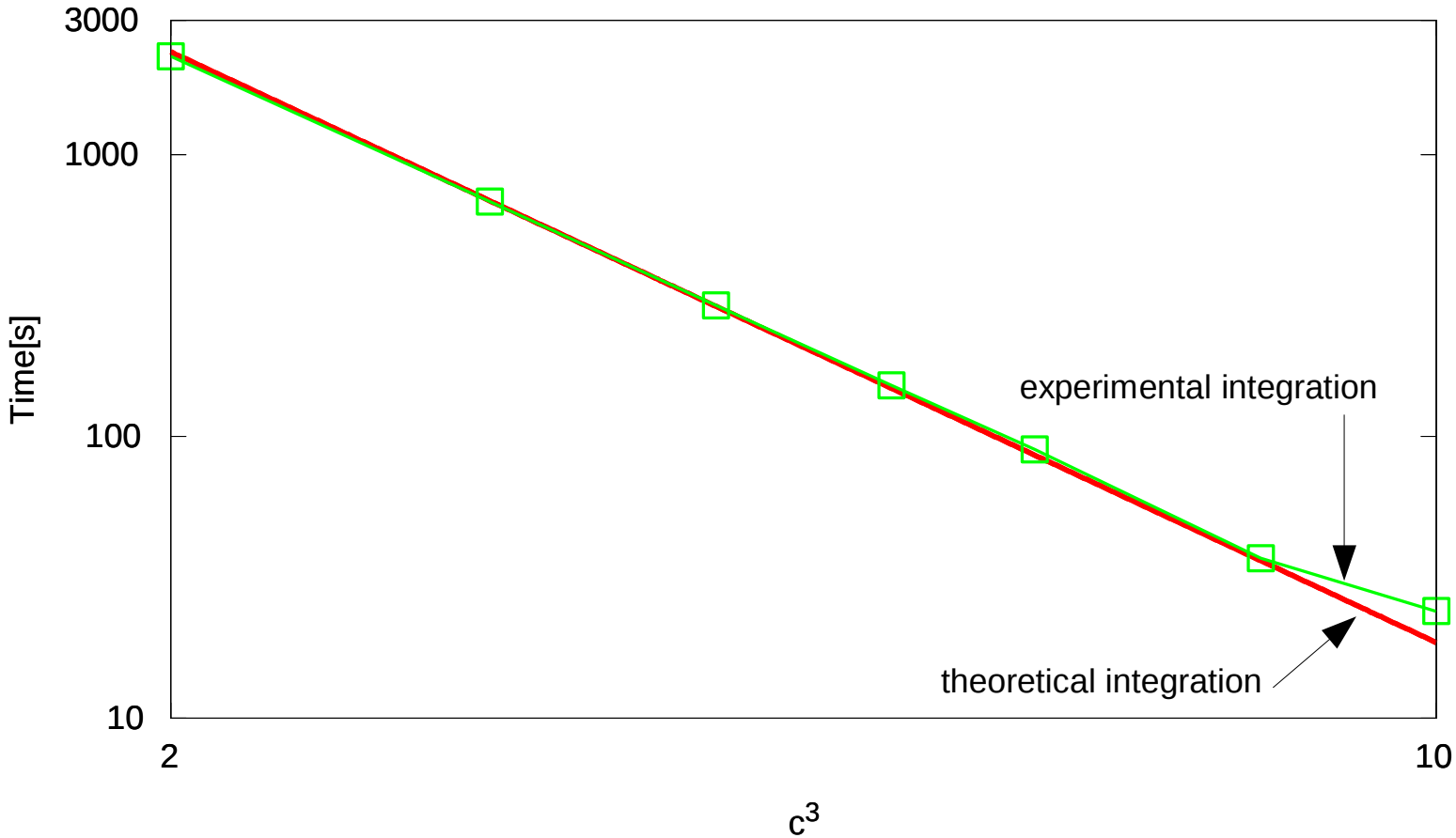


Figure: Comparison of experimental and theoretical integration time for $N = 512$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Parallel domain decomposition based explicit dynamics

The solution time takes around 1 percent of the total solver time.

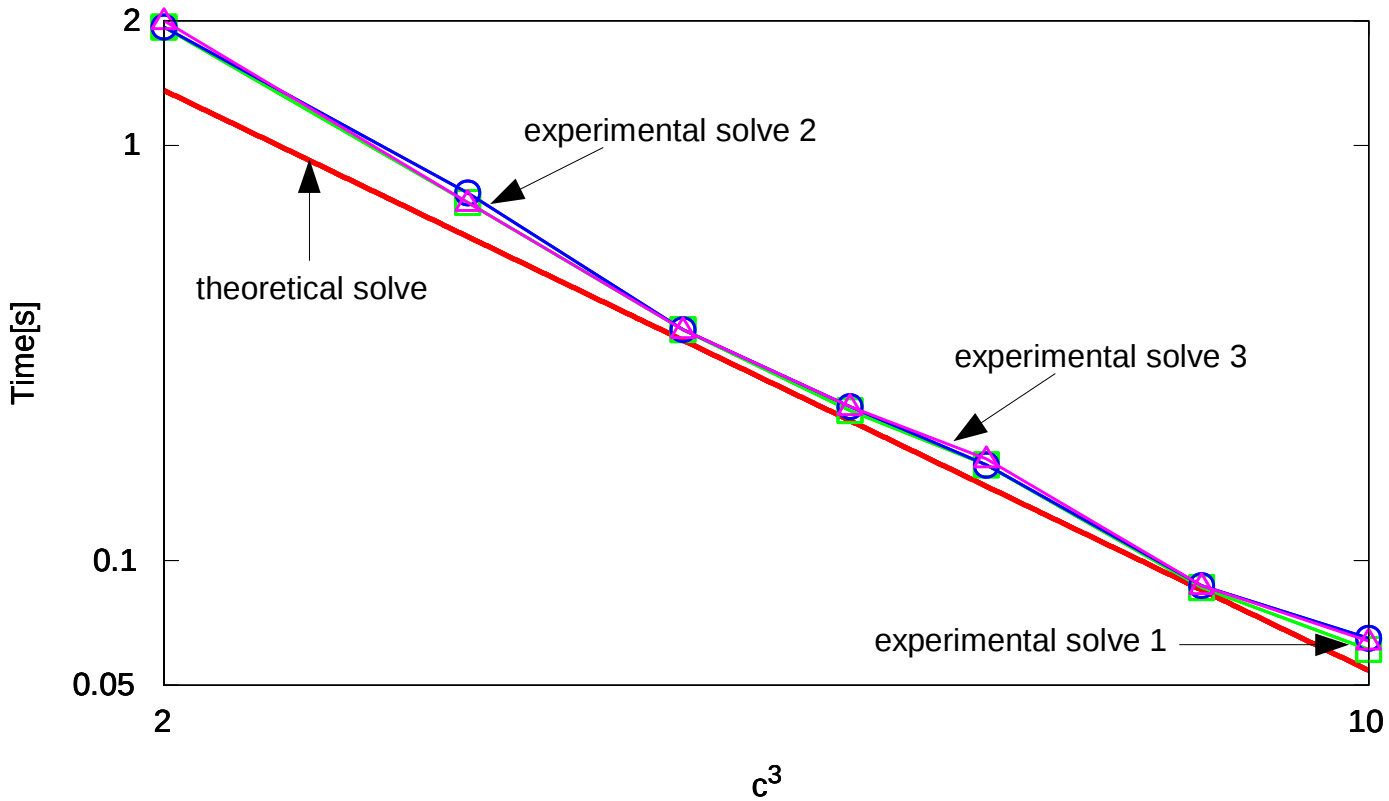


Figure: Comparison of experimental and theoretical solution times for $N = 512$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Parallel domain decomposition based explicit dynamics

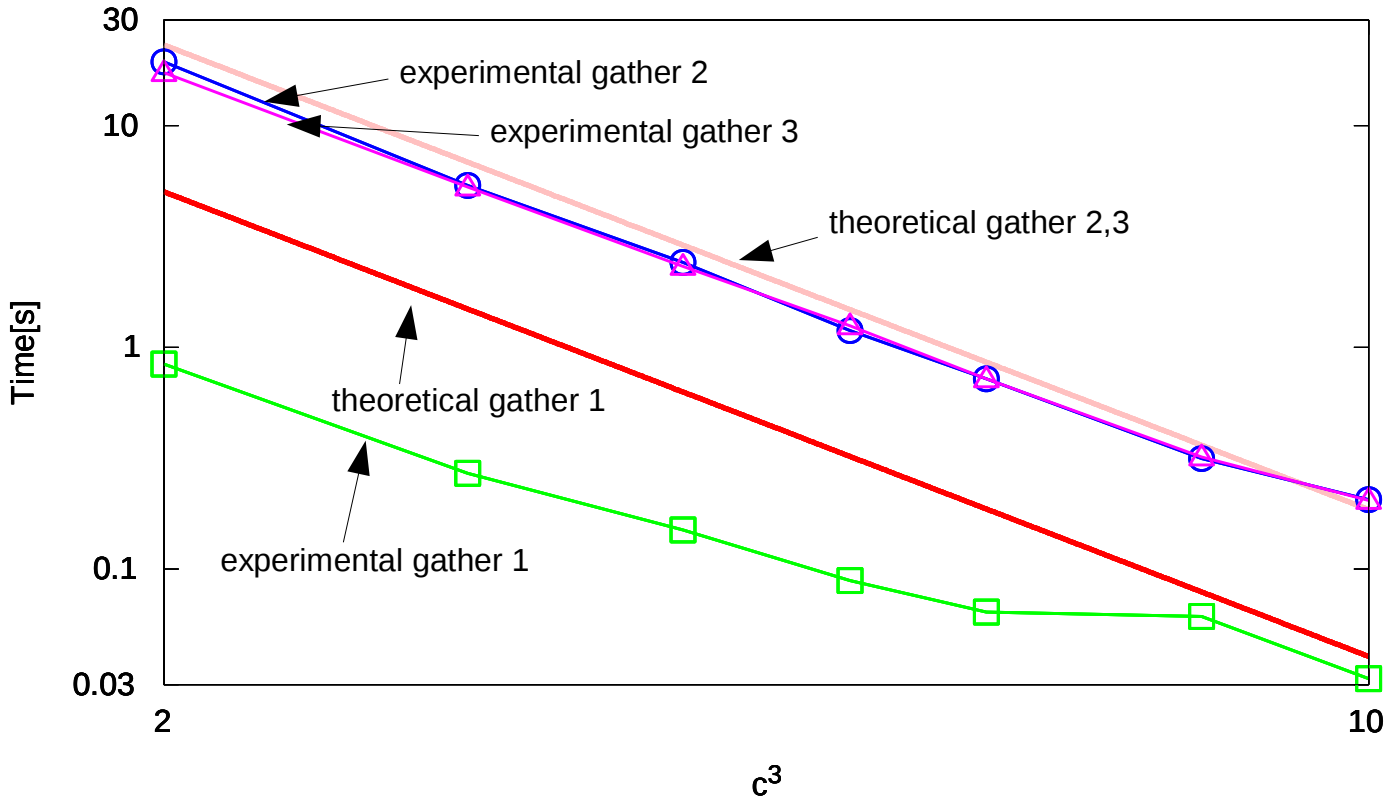


Figure: Comparison of experimental and theoretical gather times for $N = 512$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Parallel domain decomposition based explicit dynamics

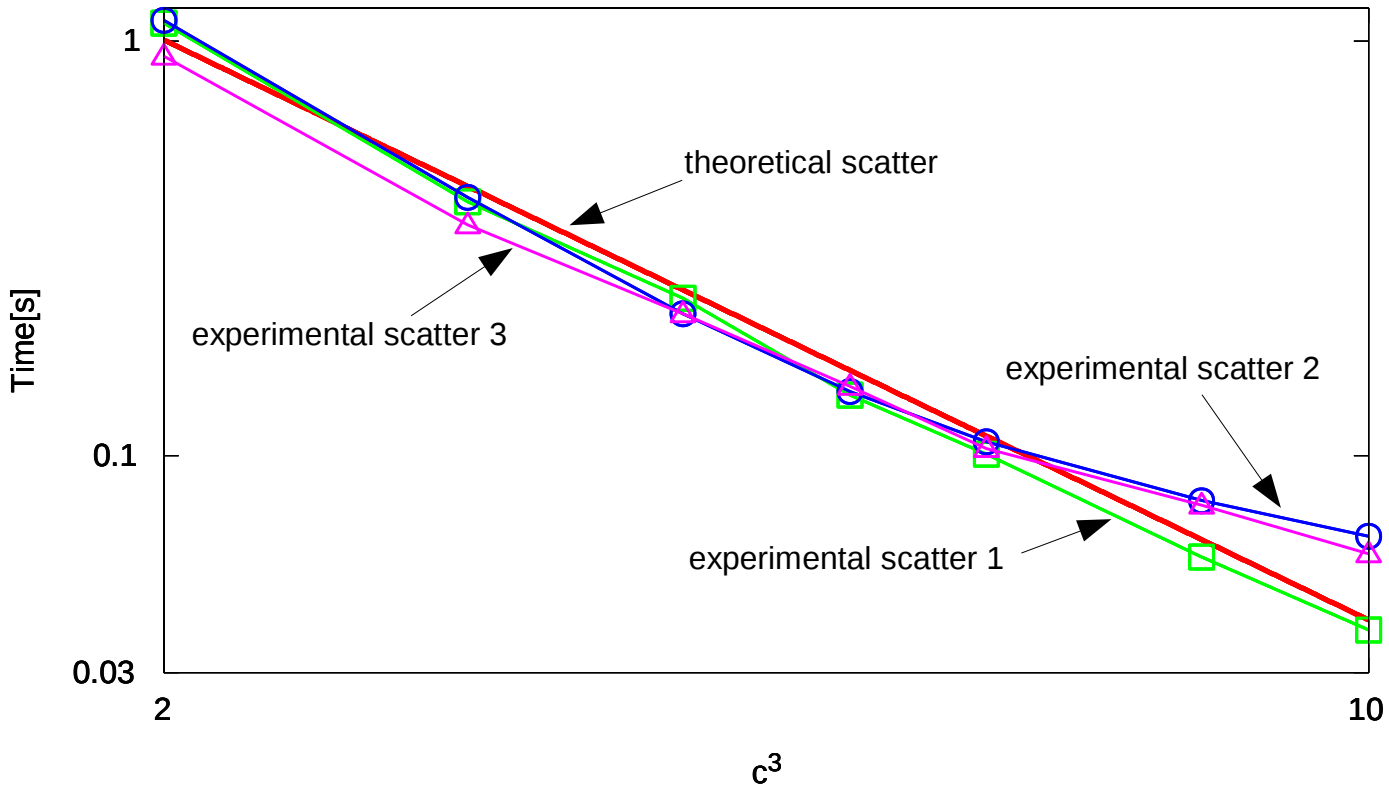


Figure: Comparison of experimental and theoretical scatter times for $N = 512$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Numerical experiments LONESTAR Linux cluster

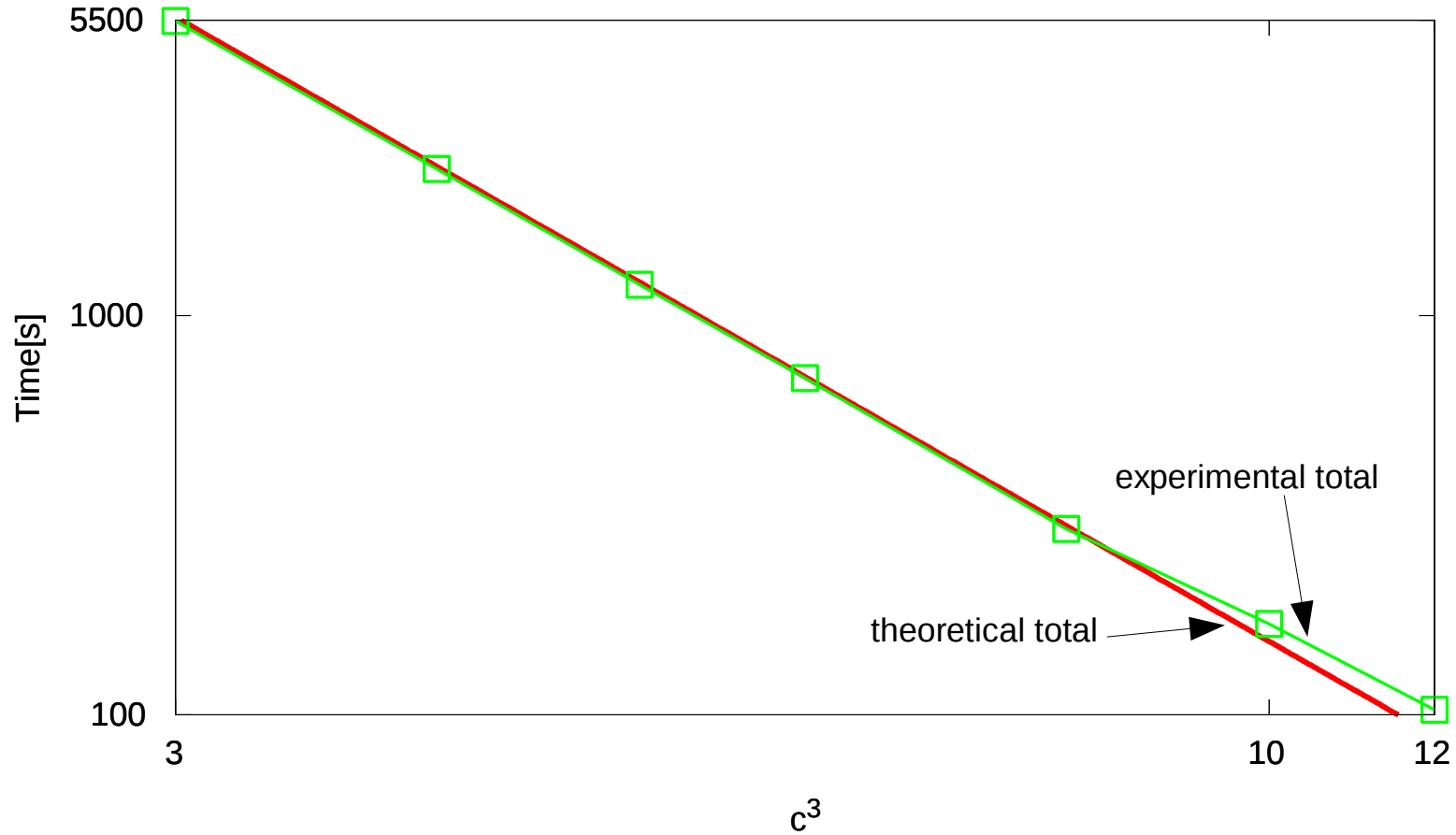


Figure: Comparison of total experimental and theoretical execution time for $N = 1024$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

The integration time is dominating the solution time significantly.

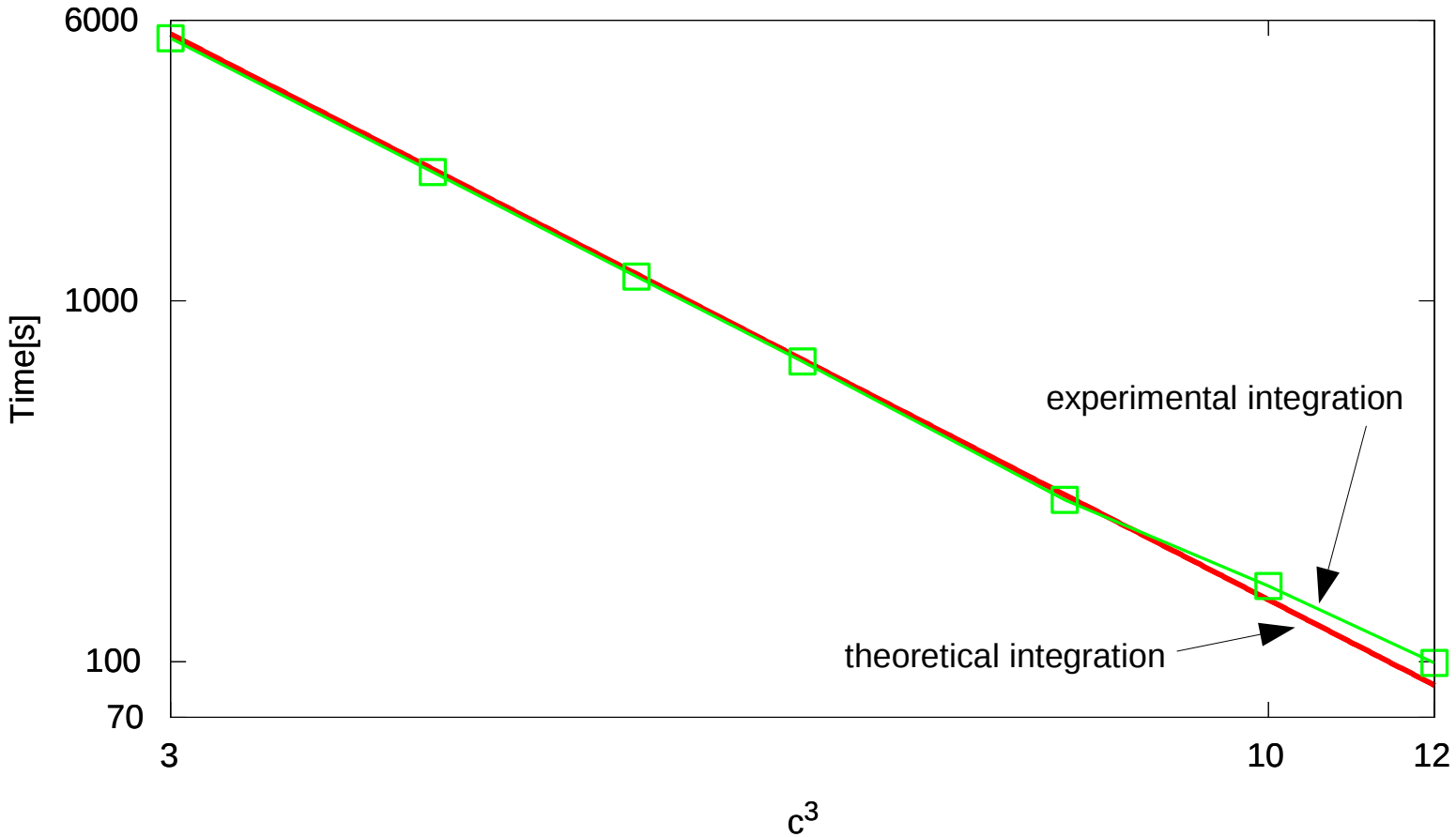


Figure: Comparison of total experimental and estimated integration time for $N = 1024$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

The solution time takes around 1 percent of the total solver time.

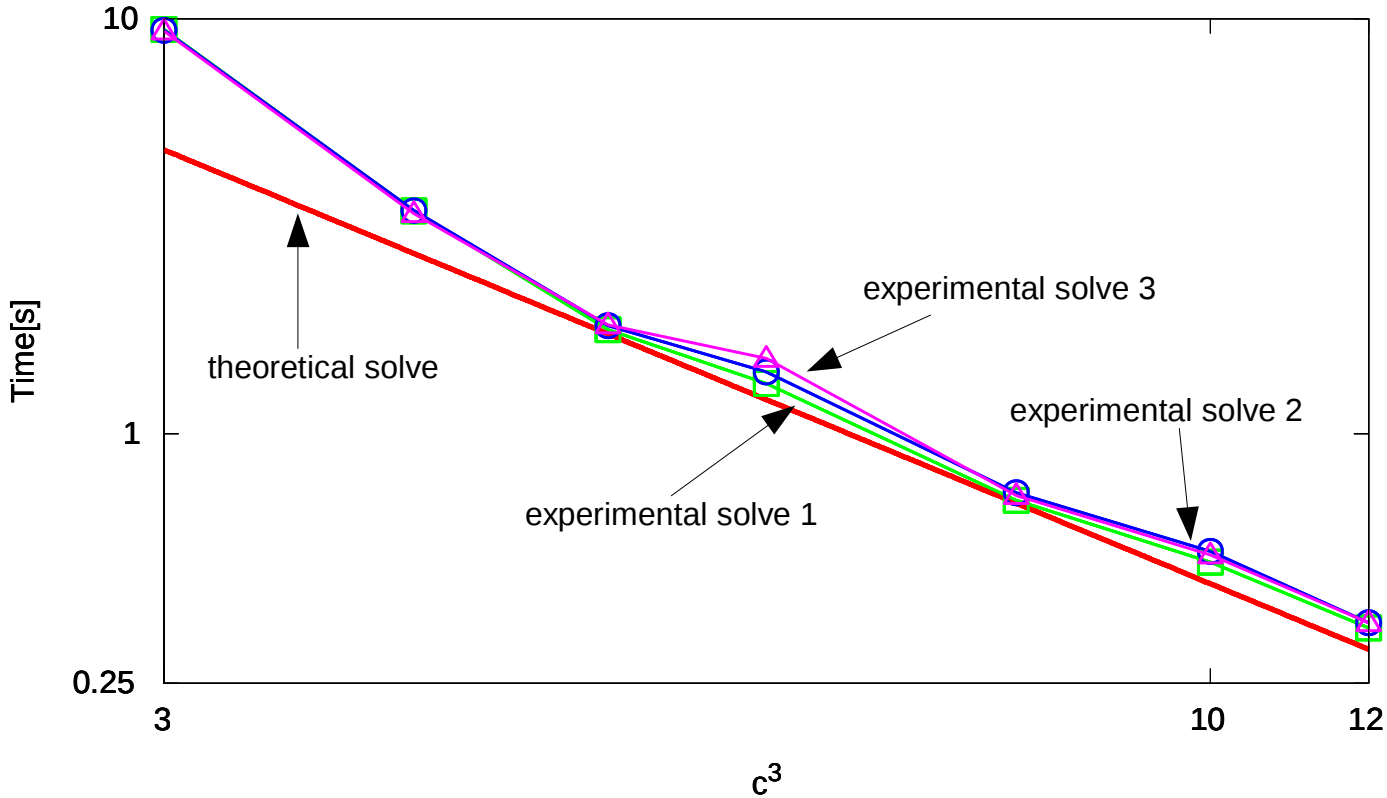


Figure: Comparison of total experimental and estimated solution times for $N = 1024$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Parallel domain decomposition based explicit dynamics

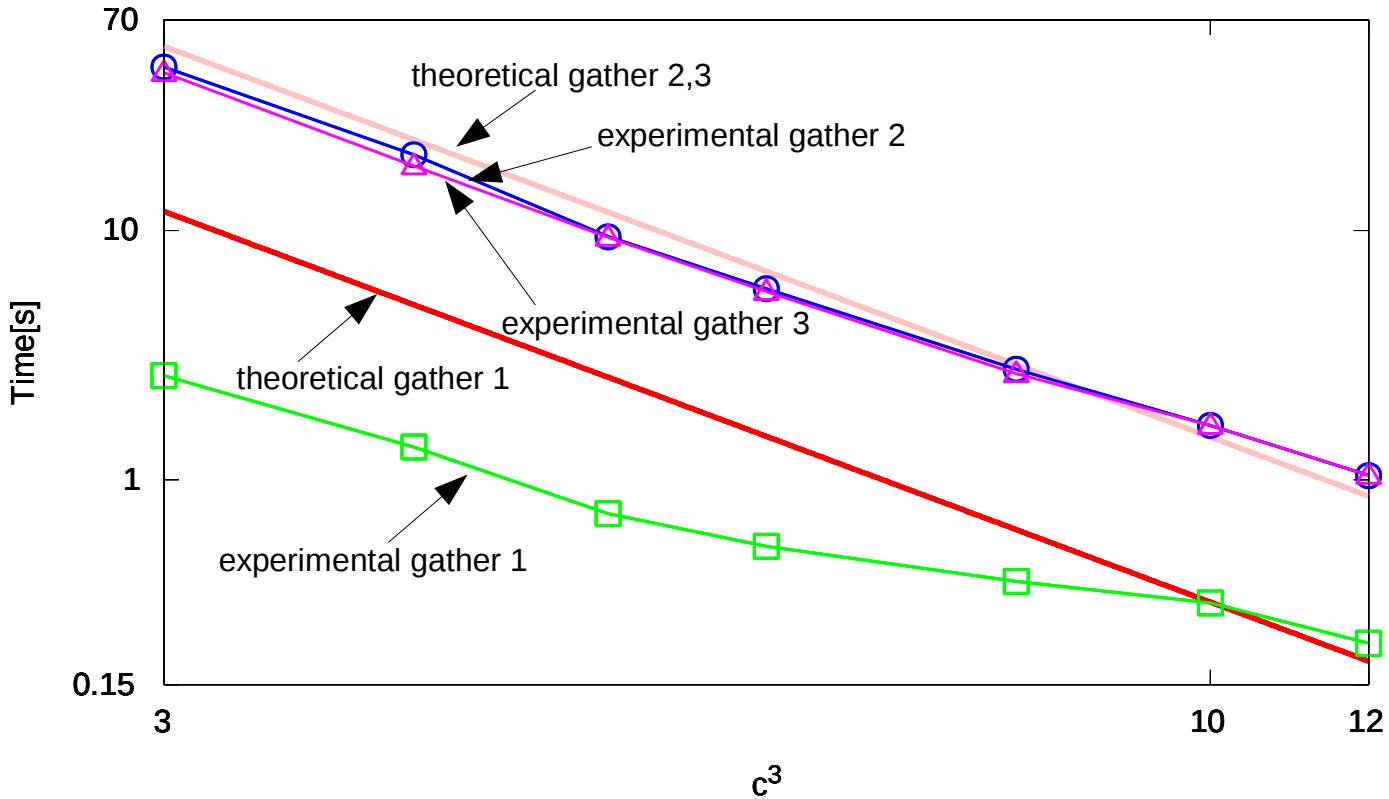


Figure: Comparison of total experimental and estimated gather times for $N = 1024$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Parallel domain decomposition based explicit dynamics

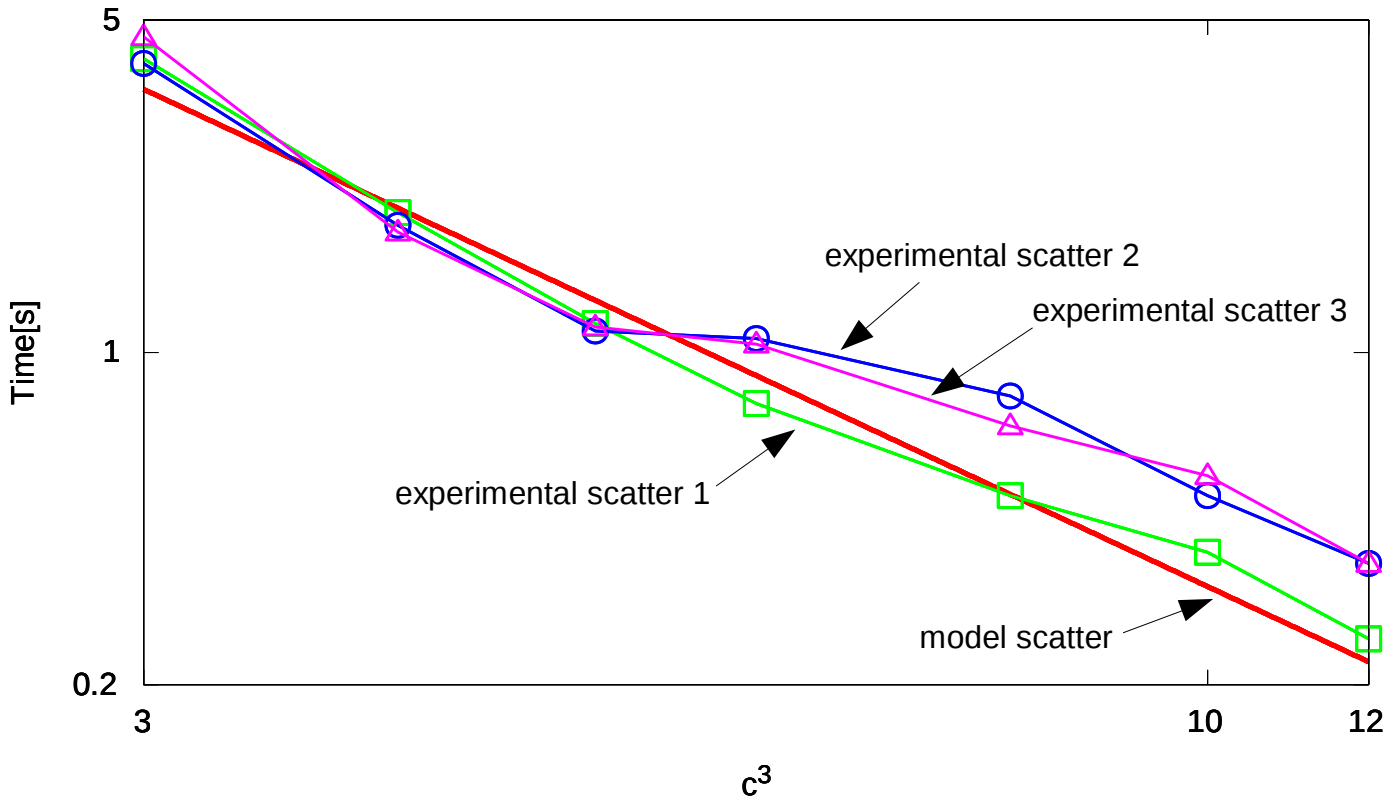


Figure: Comparison of total experimental and estimated scatter times for $N = 1024$ for $p = 3$ for different number of processors $c^3 = 2^3, \ldots, 10^3 = 8, \ldots, 1000$.

Limitations of isogeometric L2 projections

Time step size limited by Courrant-Fredrichs-Levy (CFL) condition

https://en.wikipedia.org/wiki/Courant-Friedrichs-Lewy_condition

$\frac{u_x \Delta t}{\Delta x} + \frac{u_y \Delta t}{\Delta y} \leq C_{max}$

where Δt is time step size, $\Delta x = \Delta y = h$ element size, u_x, u_y magnitude of the field, $C_{max} = 1$ for explicit method, so $(u_x + u_y) * \Delta t \leq h$

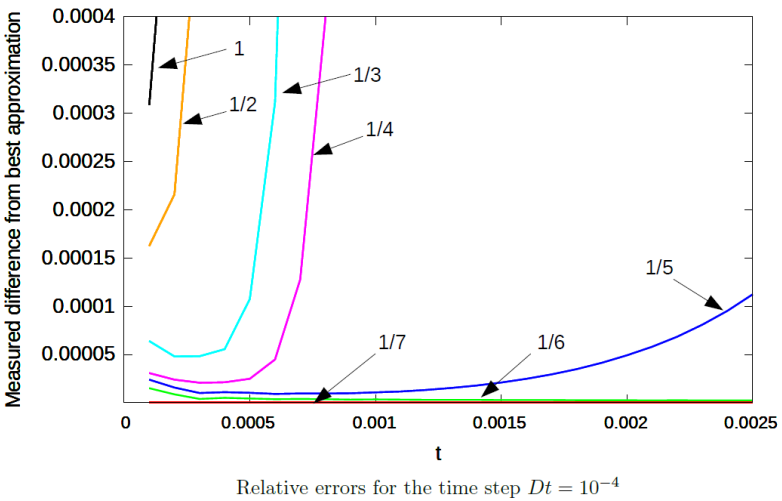


Figure: Lack of convergence for $Dt = 10^{-4}, \frac{10^{-4}}{2}, ..., \frac{10^{-4}}{5}$

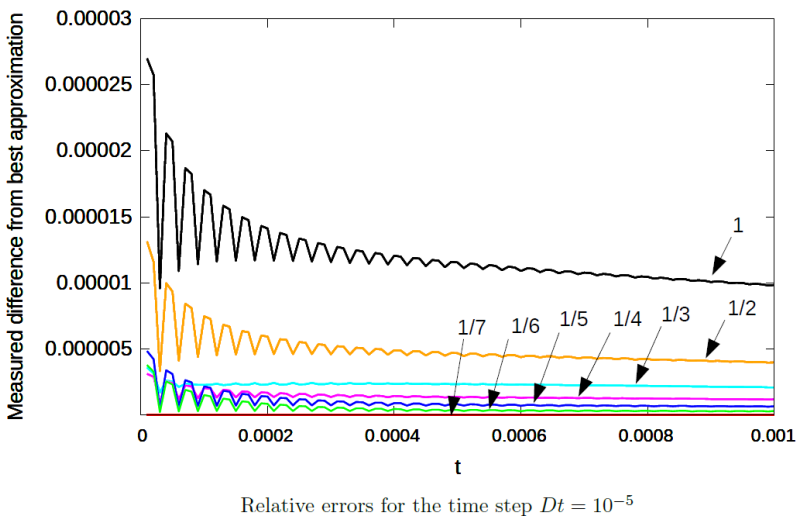


Figure: Convergence for $Dt = 10^{-5}$ and smaller time steps

Example: Too large time step for elastic wave propagation

Click in the middle

Example: Good time step for elastic wave propagation

Click in the middle

Integration of the 3D matrix

If we go for implicit method, we need to integrate the matrix

We have a mesh of $N_x \times N_y \times N_z$ elements

$nrdof = (p_x + 1)(p_y + 1)(p_z + 1)$ basis functions on each element

(p_x, p_y, p_z) denotes the B-splines order in directions x, y and z

$ngx = O(p_x)$, $ngy = O(p_y)$, $ngz = O(p_z)$ number of Gauss points

$$(B^x_{i,p} B^y_{j,p} B^z_{k,p}, B^x_{l,p} B^y_{m,p} B^z_{n,p})_{L2} =$$
$$\int_\Omega B^x_{i,p}(x) B^y_{j,p}(y) B^z_{k,p}(z) B^x_{l,p}(x) B^y_{m,p}(y) B^z_{n,p}(z) dxdydz =$$
$$\sum_E \int_E B^x_{i,p}(x) B^y_{j,p}(y) B^z_{k,p}(z) B^x_{l,p}(x) B^y_{m,p}(y) B^z_{n,p}(z) dxdydz =$$
$$\sum_E \sum_{s=1,ngx;t=1,ngy;w=1,ngz} W_s W_t W_z B^x_{i,p}(x_s) B^y_{j,p}(y_t) B^z_{k,p}(z_w) B^x_{l,p}(x_s) B^y_{m,p}(y_t) B^z_{n,p}(z_w)$$

We construct the element matrices for each element, for all the basis functions which span over the element, namely $B^x_{i,p} i = 1, p_x + 1$, $B^y_{j,p} j = 1, p_y + 1$, $B^z_{k,p} k = 1, p_z + 1$ (trial functions) and $B^x_{l,p} l = 1, p_x + 1$, $B^y_{m,p} m = 1, p_y + 1$, $B^z_{n,p} n = 1, p_z + 1$ (test functions).

Integration of the 3D matrix

We have a mesh of $N_x \times N_y \times N_z$ elements

$nrdof = (p_x + 1)(p_y + 1)(p_z + 1)$ basis functions on each element

(p_x, p_y, p_z) denotes the B-splines order in directions x, y and z

$ngx = O(p_x)$, $ngy = O(p_y)$, $ngz = O(p_z)$ number of Gauss points

- for $s = 1, ngx$ // Gauss integration points
- for $t = 1, ngy$
- for $w = 1, ngz$

  get Gauss point (x_s, y_t, z_w), weight $W_s W_t W_w$
- for $l = 1, p_x + 1$ // trial B-splines
- for $m = 1, p_y + 1$
- for $n = 1, p_z + 1$
- for $j = 1, p_x + 1$ // test B-splines
- for $j = 1, p_y + 1$
- for $k = 1, p_z + 1$
- aggregate

  $W * B^x_{i,p}(x_s) B^y_{j,p}(x_t) B^z_{k,p}(x_w), B^x_{l,p}(x_s) B^y_{m,p}(x_t) B^z_{n,p}(x_w)$

$p_x = p_y = p_z = p$ computational complexity $O(p^9)$, if p=9 it is 10^9

Sequential integration of the 3D matrix

$F = 0.d0$

```
do ex = 1,nelemx //Loop through elements
  do ey = 1,nelemy
    do ez = 1,nelemz
      J = Jx(ex)*Jy(ey)*Jz(ez) //element Jacobian
      do kx = 1,ngx //Loop through Gauss points
        do ky = 1,ngy
          do kz = 1,ngz
            W = Wx(kx)*Wy(ky)*Wz(kz) //Gauss weight
            value = fvalue(Xx(kx,ex),Xy(ky,ey),Xz(kz,ez))
            do ax = 0,px //B-splines
              do ay = 0,py
                do az = 0,pz
            do bx = 0,px //B-splines
              do by = 0,py
                do bz = 0,pz
          call compute_index(ind,ax,ay,az,ex,ey,ez,nx,ny,nz)
          call compute_index(ind1,bx,by,bz,ex,ey,ez,nx,ny,nz)
               A(ind,ind1) = A(ind,ind1) +
  NNx(0,ax,kx,ex)*NNy(0,ay,ky,ey)*NNz(0,az,kz ,ez)*
  NNx(0,bx,kx,ex)*NNy(0,by,ky,ey)*NNz(0,bz,kz ,ez)*J*W*value
```

Parallel OpenMP integration of the 3D matrix

OpenMP = Open Multi-Processing

```
!$OMP PARALLEL DO
!$OMP& DEFAULT(SHARED)
!$OMP& FIRSTPRIVATE
    (iy,ex,ey,ez,J,kx,ky,kz,W,value,ax,ay,az,bx,by,bz,ind,ind1)
!$OMP& REDUCTION(+:nr_nonzeros)
  do iy=1,miy //Now it is 1 loop over elements
   call map_indexes(iy,ex,ey,ez)
   J = Jx(ex)*Jy(ey)*Jz(ez) //element Jacobian
   do kx = 0,ngx //loop through Gauss points
    do ky = 0,ngy
     do kz = 0,ngz
         W = Wx(kx)*Wy(ky)*Wz(kz) //Gauss weight
         value = fvalue(Xx(kx,ex),Xy(ky,ey),Xz(kz,ez))
         do ax = 0,px //trail B-splines along x,y,z
          do ay = 0,py
           do az = 0,pz
         do bx = 0,px //test B-splines along x,y,z
          do by = 0,py
           do bz = 0,pz
            call compute_index(ind,ax,ay,az,ex,ey,ez,nx,ny,nz)
            call compute_index(ind1,bx,by,bz,ex,ey,ez,nx,ny,nz)
            A(ind,ind1) = A(ind,ind1) +
  NNx(0,ax,kx,ex)*NNy(0,ay,ky,ey)*NNz(0,az,kz ,ez)*
  NNx(0,bx,kx,ex)*NNy(0,by,ky,ey)*NNz(0,bz,kz ,ez)*J*W*value
!$OMP END PARALLEL DO
```

Numerical results

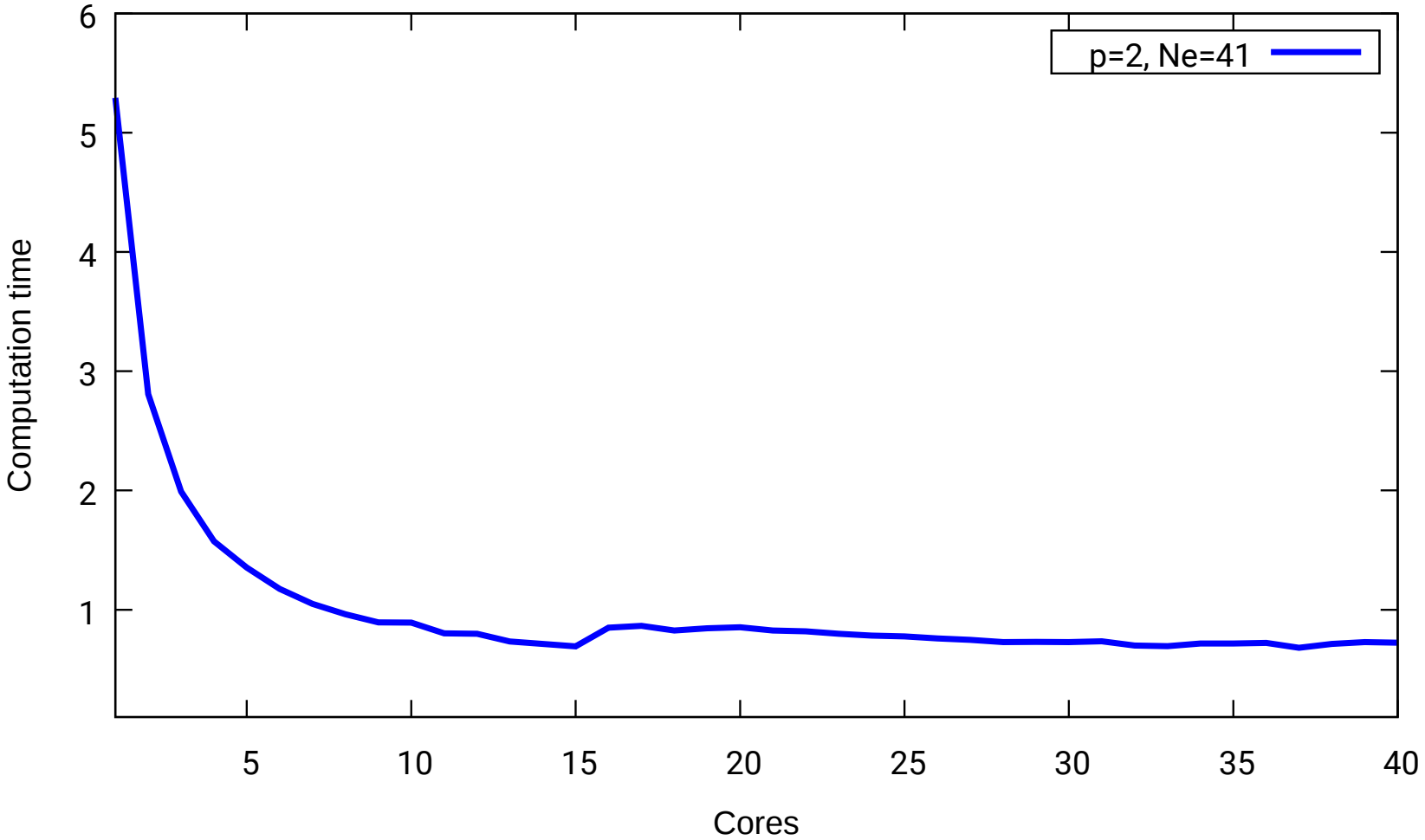


Figure: Execution time of the parallel integration algorithm, when increasing number of cores. 3D element with quadratic polynomials

Numerical results

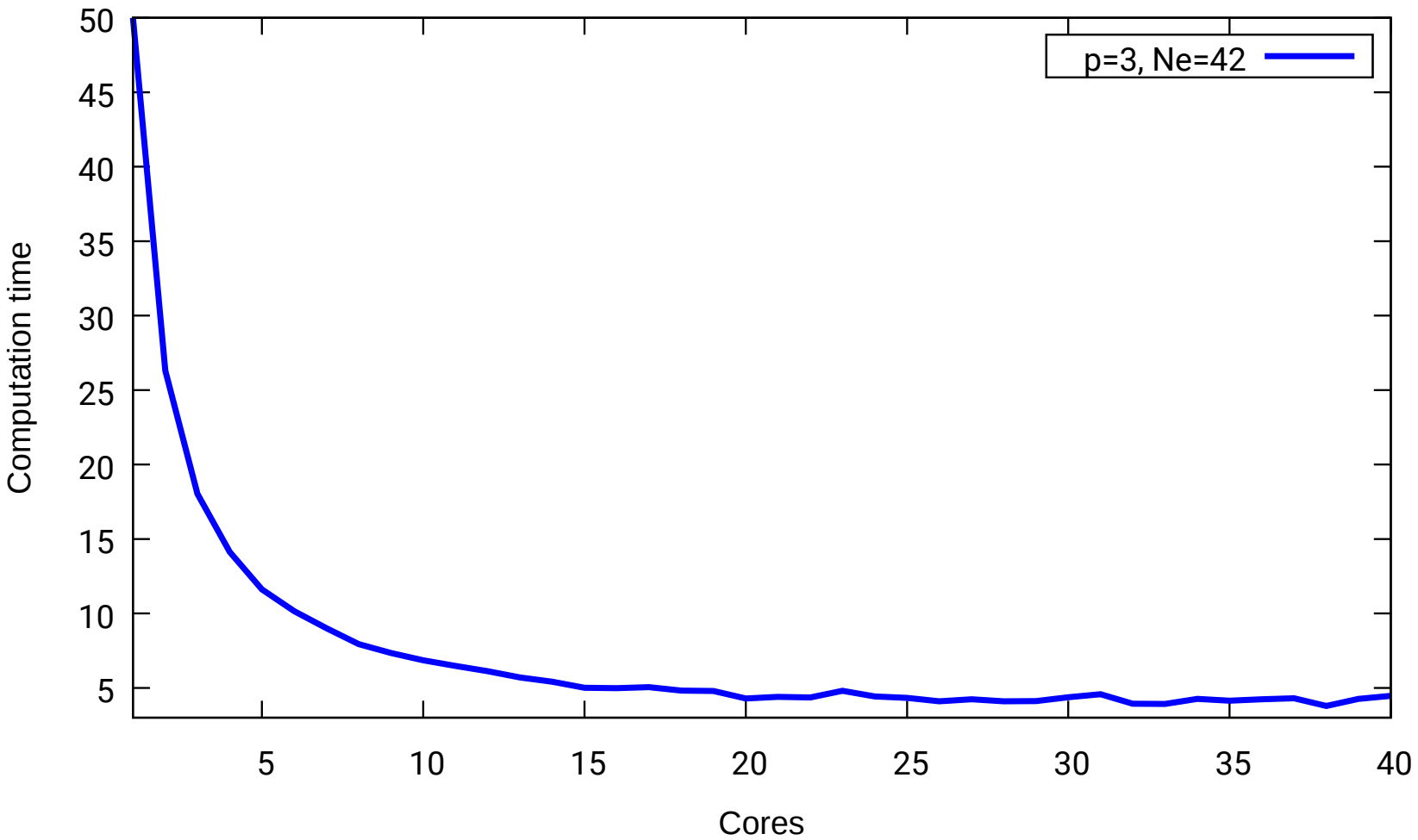


Figure: Execution time of the parallel integration algorithm, when increasing number of cores. 3D element with cubic polynomials

Numerical results

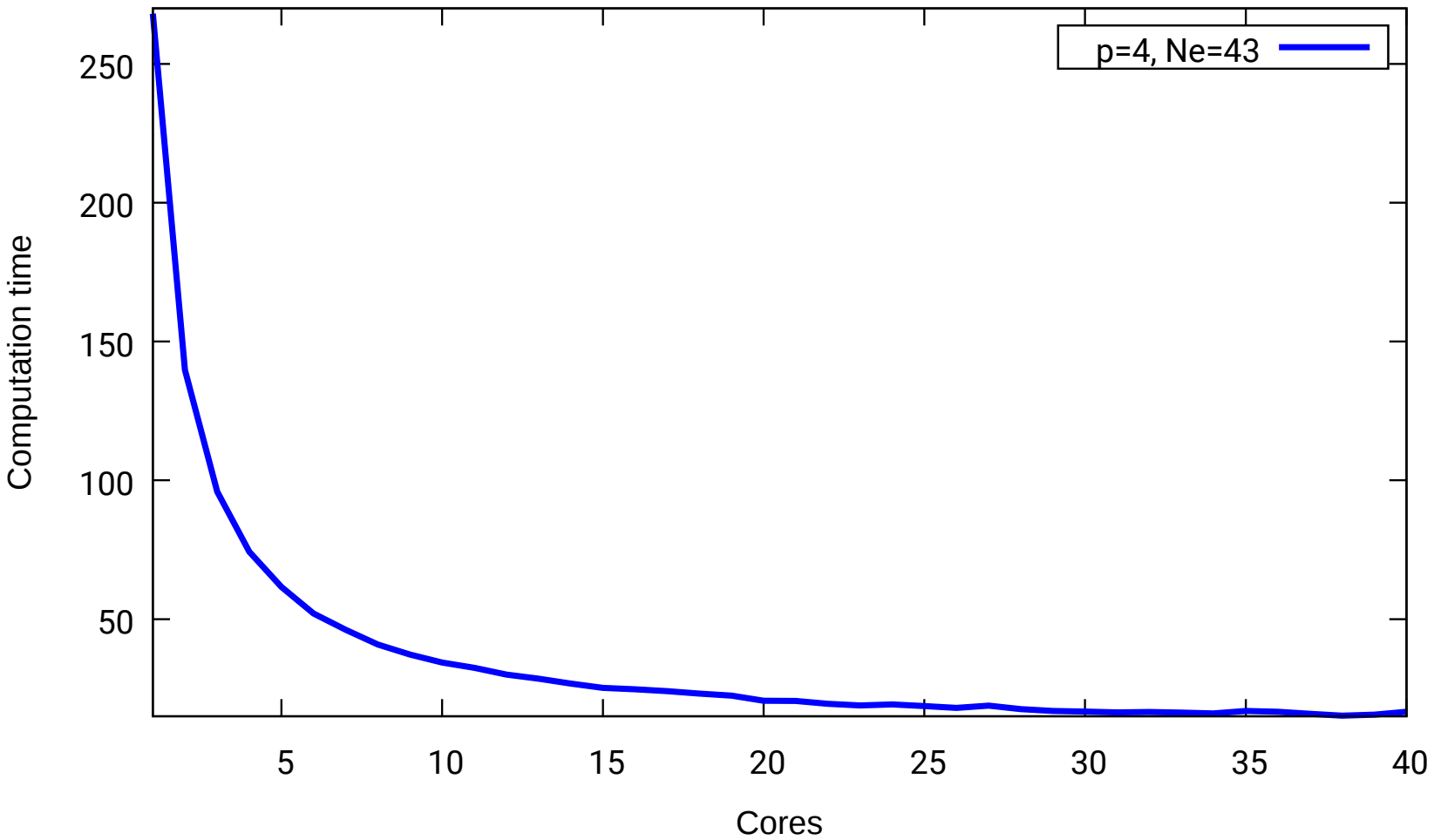


Figure: Execution time of the parallel integration algorithm, when increasing number of cores. 3D element with quartic polynomials

Numerical results

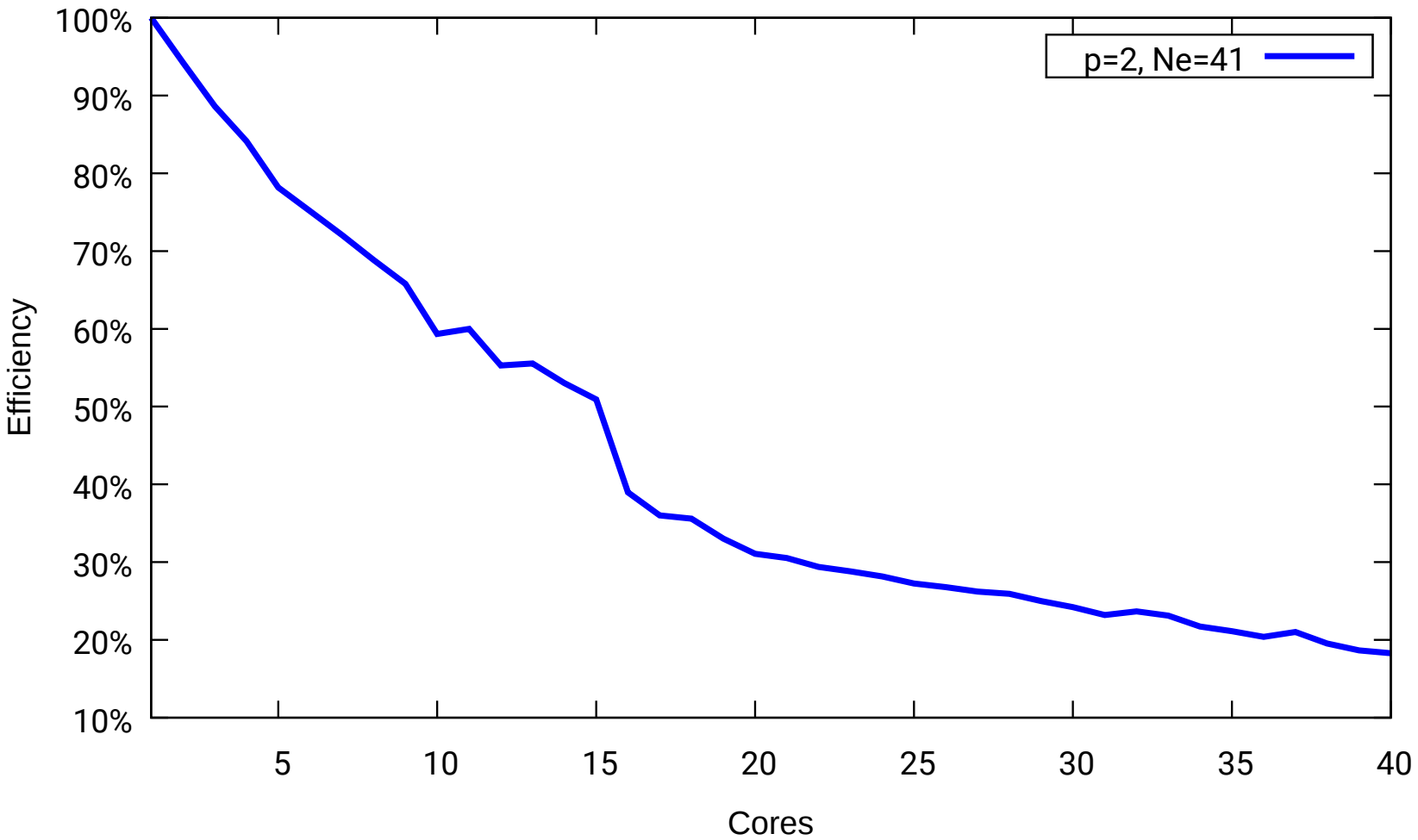


Figure: Efficiency of the parallel integration algorithm, when increasing number of cores. 3D element with quadratic polynomials

Numerical results

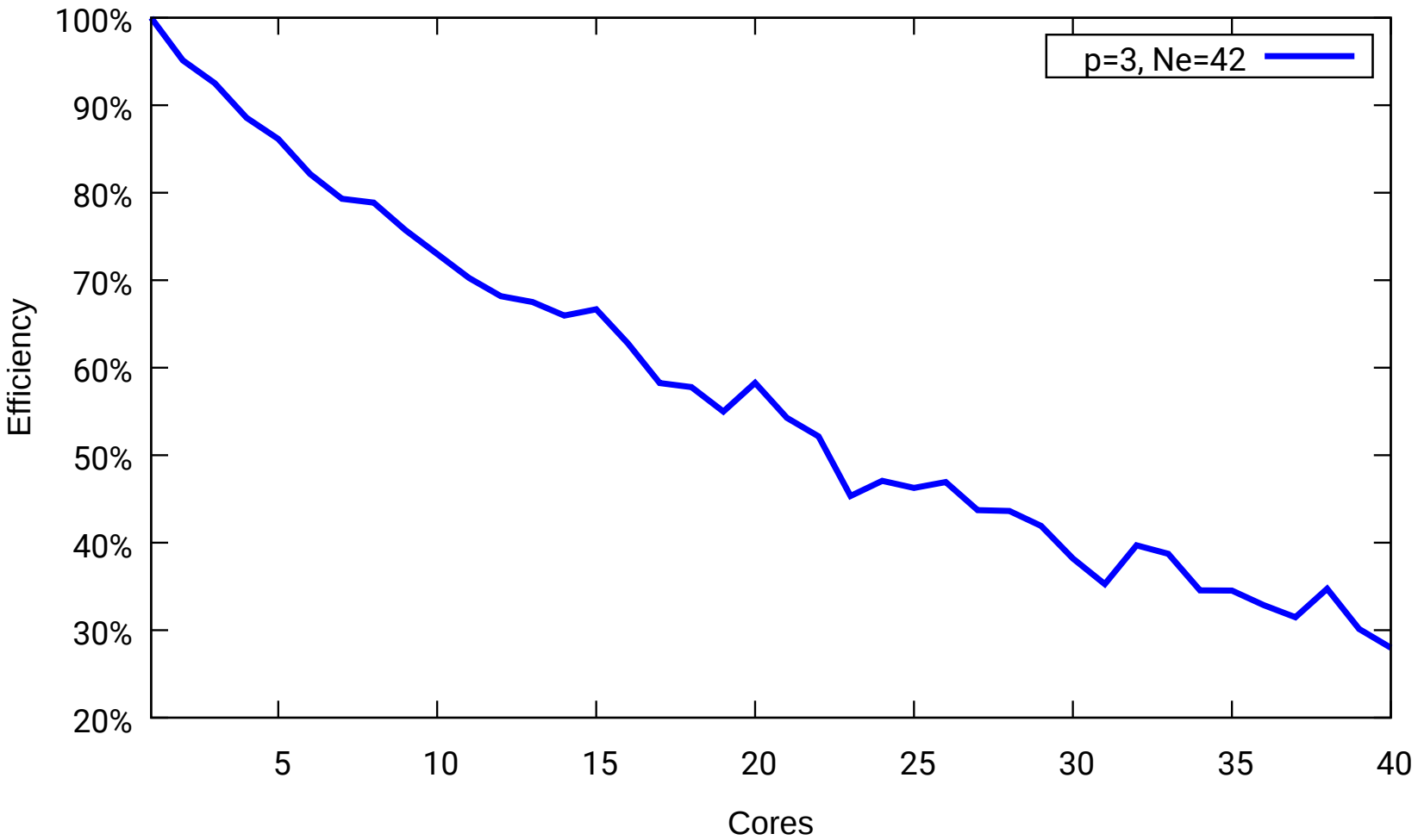


Figure: Efficiency of the parallel integration algorithm, when increasing number of cores. 3D element with cubic polynomials

Numerical results

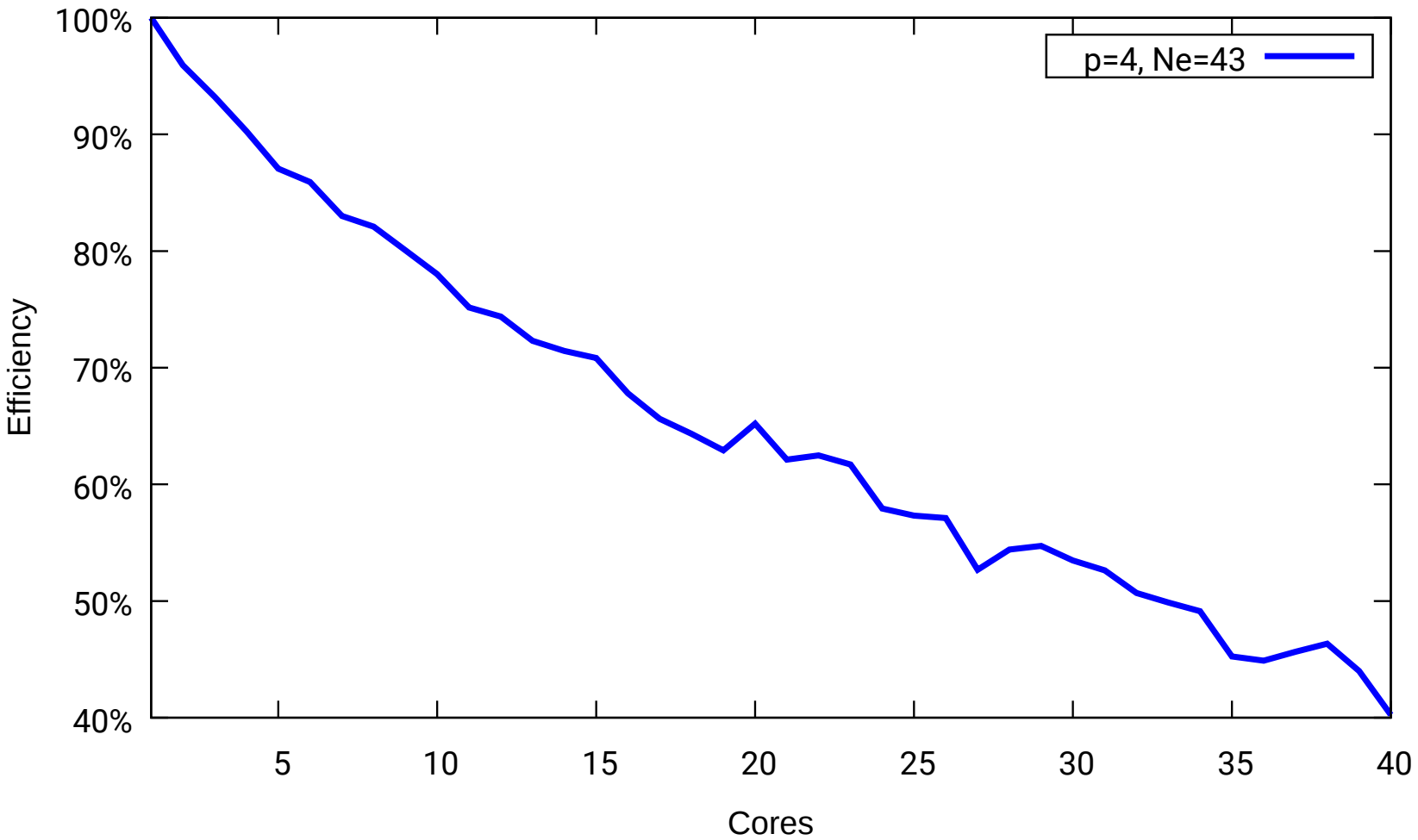


Figure: Efficiency of the parallel integration algorithm, when increasing number of cores. 3D element with quartic polynomials

Numerical results

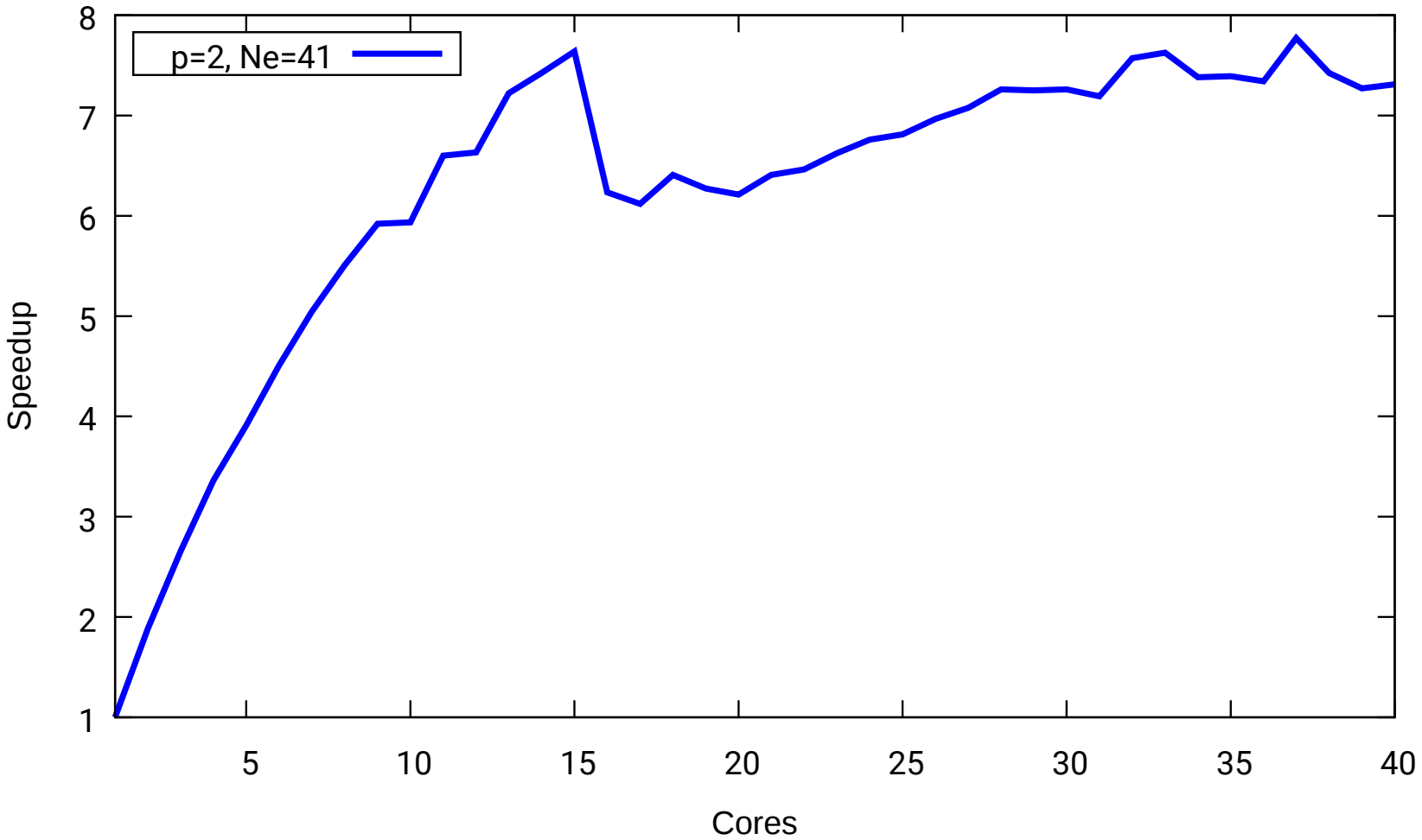


Figure: Speedup of the parallel integration algorithm, when increasing number of cores. 3D element with quadratic polynomials

Numerical results

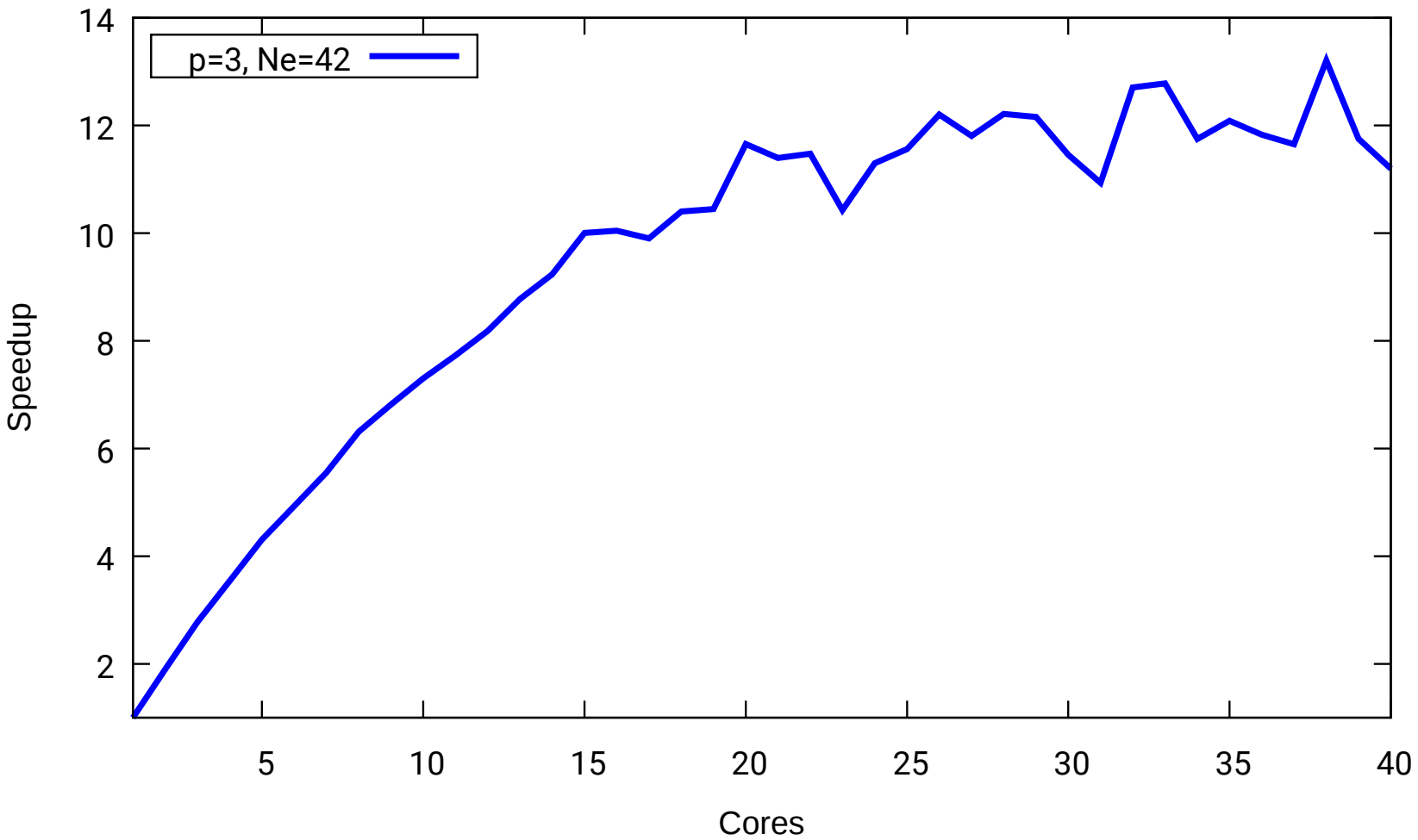


Figure: Speedup of the parallel integration algorithm, when increasing number of cores. 3D element with cubic polynomials

Numerical results

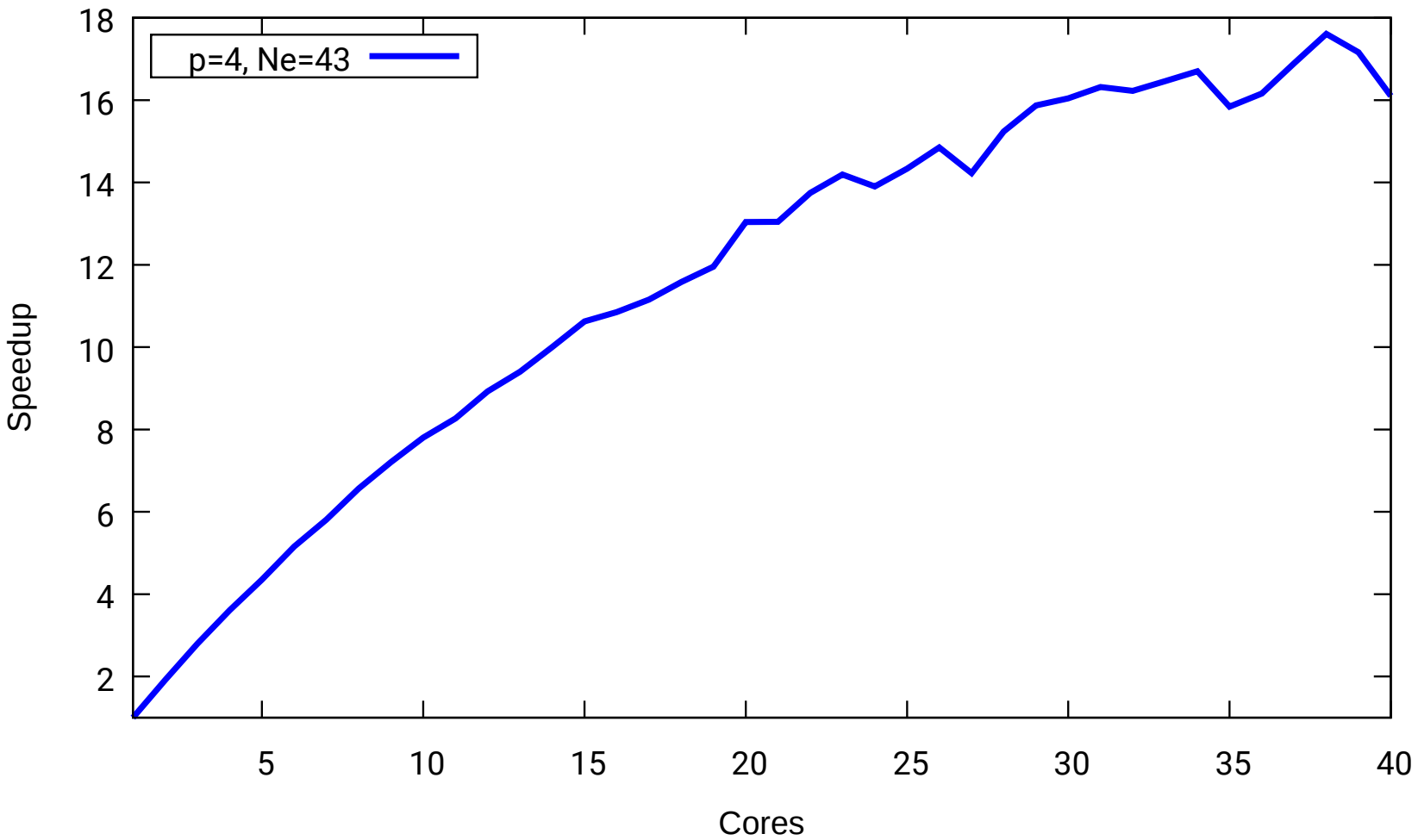


Figure: Speedup of the parallel integration algorithm, when increasing number of cores. 3D element with quartic polynomials