Revisiting exponential integrator methods for HPC with a mini-application

Douglas Shanks, HPC group, AWE plc

COPYRIGHT
This publication is **UK Ministry of Defence © Crown copyright** (2020). Material and information contained in this publication may be reproduced, stored in a retrieval system and transmitted for MOD use only, except where authority for use by other organisations or individuals has been authorised by a Patent Officer of the Defence Intellectual Property Rights (DIPR) whose details appear below.
Crown Copyright and Merchandise Licensing
Defence Intellectual Property rights
MOD Abbey Wood South
Poplar 2 #2214
Bristol BS34 8JH
Email: DIPR-Copyright@mod.gov.uk

11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA20)

November 12, 2020

Overview

- Introduction
  - HPC at AWE
  - Motivation
  - Review of communication avoiding methods
- Methods and implementation
  - TeaLeaf mini-app
  - Exponential Integrators
  - Implementation
  - Test problem
- Numerical Results
  - Matrix powers kernel
  - Strong scaling and efficiency
- Summary

HPC at AWE

Current hardware:

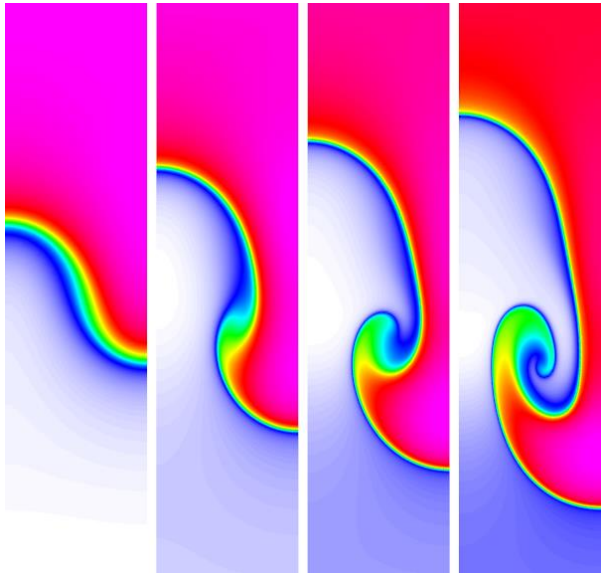
- Vulcan – **7.4 PFLOPS** with **160,000 AMD EPYC Rome cores.**
- Damson – **4.3 PFLOPS** with **116,640 Intel Broadwell cores.**
- Rosewood – **1.9 PFLOPS** with **48,284 Intel Haswell cores.**
- Spruce – **1.8 PFLOPS** with **80,160 Intel Ivy Bridge cores.**

HPC is a critical element of the CTBT method:

- Certification.
- Hydrodynamics & Plasma experiments.
- Engineering parametric analysis.
- Confirmation of experiments -> UGT data + Integrated experiments.
- Physics codes + Materials theory -> HPC (simulation codes)

Motivation

- Wide range of Physics being solved in production codes.
  - Hydrodynamics.
  - Thermal conduction.
  - Neutron/X-Ray transport.
- AWE mainly use implicit timestepping, so require:
  - Linear solver (CG, GMRES, BiCGSTAB).
  - Preconditioner (Block Jacobi, GMG, AMG).
  - Through PETSc or Hypre.
- However scalability issues exist for different solvers options:
  - Single level scale poorly with iterations and at scale.
  - Multilevel **can** scale poorly in parallel (setup, communications).



Review of some Communication Avoiding Methods

- **Polynomial based methods** [O'Leary (91), Loe et al (19)] :
  - Precondition Krylov method (CG, GMRES) reducing iterations.
  - Hence reduction in global synchronisations (dot products).
  - Comes at cost of extra SpMVs and AXPYs.
- **Matrix Powers Kernel (MPK)** [Demmel et al (07)]**:**
  - Allows reduction in local communication by artificially increasing halo depth.
  - Useful when algorithm involves lots of SpMVs.
  - Comes at cost of increased storage requirement.

Relation to current and previous work

- **CPPCG** [Cluster 2017]**:**
  - Implementation of a polynomial preconditioned CG method with MPK.
  - Computes Chebyshev polynomial to precondition CG.
  - Still possible to combine with other preconditioners.
- **CPEXI:**
  - Implementation of a polynomial approximation of Exponential Integrators with MPK.
  - Uses Chebyshev series to approximate matrix exponential.
  - Order of magnitude faster than PPCG in our experiments.

TeaLeaf mini-app

Solving linear heat diffusion IBVP on $\Omega \subset \mathbb{R}^{2,3}$.

$$\frac{\partial u}{\partial t} = \nabla.(\alpha \nabla \mathrm{u}), \text{in } \Omega \times (0,\mathrm{T}]$$
$$\alpha \frac{\partial u}{\partial n} = 0, \text{on } \partial\Omega \times [0, \mathrm{T}]$$
$$u = u_0, \text{in } \Omega \text{ at } \mathrm{T} = 0$$

- 2nd order finite difference in space and backward Euler in time.
- Solve following linear system at each timestep

$$AU^{n+1} = U^n$$

- Can be found at https://github.com/UK-MAC/TeaLeaf.
- Part of Mantevo project http://mantevo.org.

Exponential Integrators

- Discretise our IBVP with respect to space.

$$\frac{\partial u}{\partial t} = Lu.$$

- Then realise as an exponential time differencing scheme

$$u_{n+1} = \phi_o(\Delta t L)u_n.$$

- We choose to approximate ϕ_o using a truncated Chebyshev series [Suhov (2014)] .

$$\phi_o^M(z) = \sum_{j=0}^{M} \alpha_j e^{-R} I_j(R) T_j(R^{-1}z + 1)$$

  - $\alpha_0 = 1, \alpha_j = 2\ \forall j \geq 1.$
  - $R = \frac{|\lambda_{min}|}{2},\ M \approx (-2R\log(\epsilon/2))^{1/2}$ for error ϵ.
  - $I_j =$ modified Bessel Functions.
  - $T_j =$ Chebyshev polynomials.

PEXI implementation

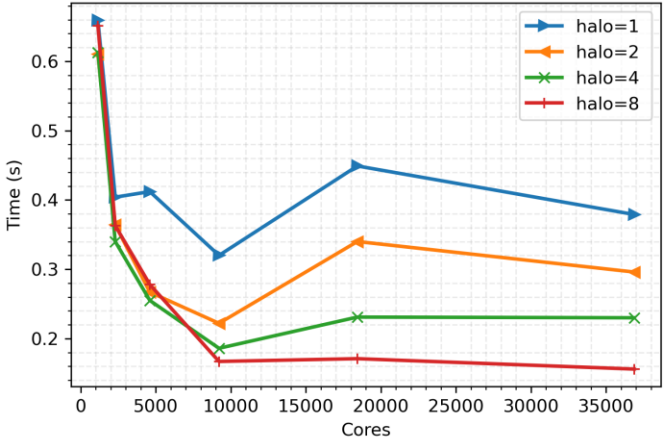
- Implemented Exponential Forward Euler method.
  - TOMS644 for Bessel functions.
  - Used both handwritten kernels and PETSc.
- Using TeaLeaf reference as the base.
  - Matrix-free Kernels written for Chebyshev recurrence.
  - Matrix powers kernel (MPK) as used in native CPPCG method.
  - Kernels written in OpenMP and CUDA
- Using TeaLeaf PETSc as the base.
  - Utilise DM grid structure.
  - Implement Chebyshev recurrence using PETSc's MatVec operations.
  - No MPK.

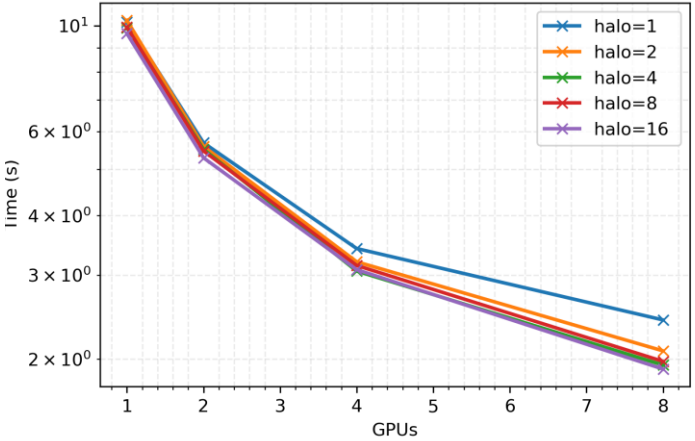
```
div_pexi_R = 2.0_8/pexi_R
BN = bessel_coeffs(n)

!$OMP PARALLEL PRIVATE(j,k) SHARED(div_pexi_R, BN)
!$OMP DO
DO k=y_min_bound,y_max_bound
    DO j=x_min_bound,x_max_bound
        utemp(j,k) = div_pexi_R((-rx*(Kx(j,k)+Kx(j+1,k))&
                    -ry*(Ky(j,k)+Ky(j,k+1))*sd(j,k)&
                    +rx*(Kx(j+1,k)*sd(j+1,k)&
                    +Kx(j,k)*sd(j-1,k))&
                    +ry*(Ky(j,k+1)*sd(j,k+1)&
                    +Ky(j,k)*sd(j,k-1))&
                    +2.0_8*sd(j,k)-rtemp(j,k)
        u(j,k) = u(j,k) + 2.0_8*BN*utemp(j,k)
    ENDDO
ENDDO
!$OMP END DO
!$OMP END DO NOWAIT
!$OMP DO
DO k=y_min_bound,y_max_bound
    DO j=x_min_bound,x_max_bound
        rtemp(j, k) = sd(j, k)
        sd(j,k) = utemp(j,k)
    ENDDO
ENDDO
!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

Setup for experiments

- Ran Benchmark 5 (crooked pipe) from TeaLeaf repo:
  - 4000^2 grid size.
  - 10 timesteps with $\Delta t = 0.004$.
  - Condition number of 5.1×10^4.
  - Linear solvers ran to residual of 1.0×10^{-15}.
  - PEXI methods ran to error of 1.0×10^{-5}.
- Ran on AWE's Damson and Bullace machines:
  - Intel Broadwell architecture.
  - Used Intel 18.0 compiler and MPI.
  - CUDA 10.2 used for V100 GPU calculations.
- Third party libraries used:
  - PETSc 3.8.0
  - Hypre 2.13

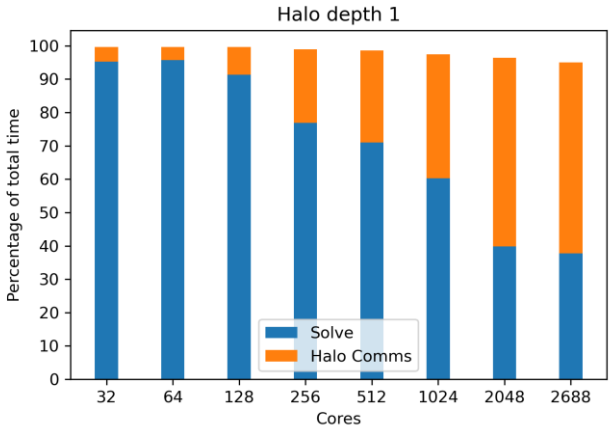
Matrix Powers Kernel – Strong Scaling

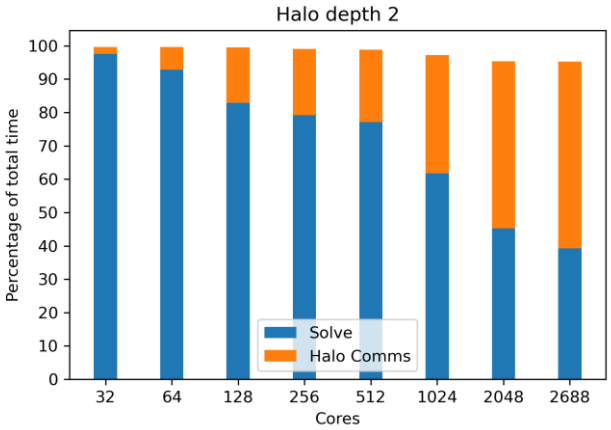


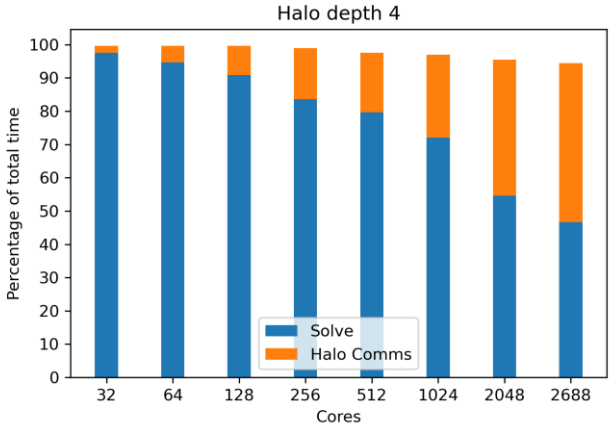


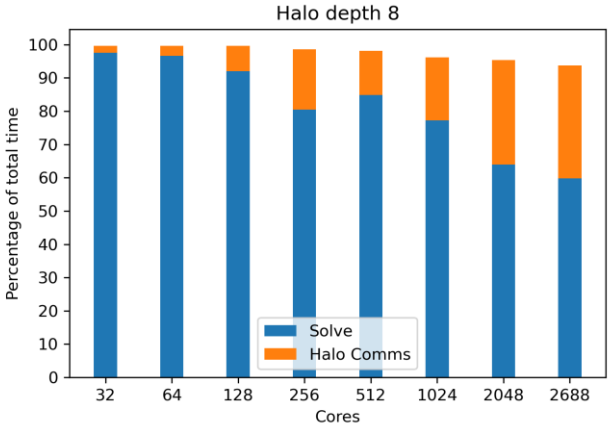
- CPU (1152 – 36864 cores) :
  - MPK most effective beyond 1152 cores with halo depth 8 giving best time.
- GPU (1 – 8 GPUs):
  - Halo depth of 4 or greater gives best performance.
  - Access to more GPUs would hopefully show similar gains to CPU.

Matrix Powers Kernel – Solve vs Halo comms time

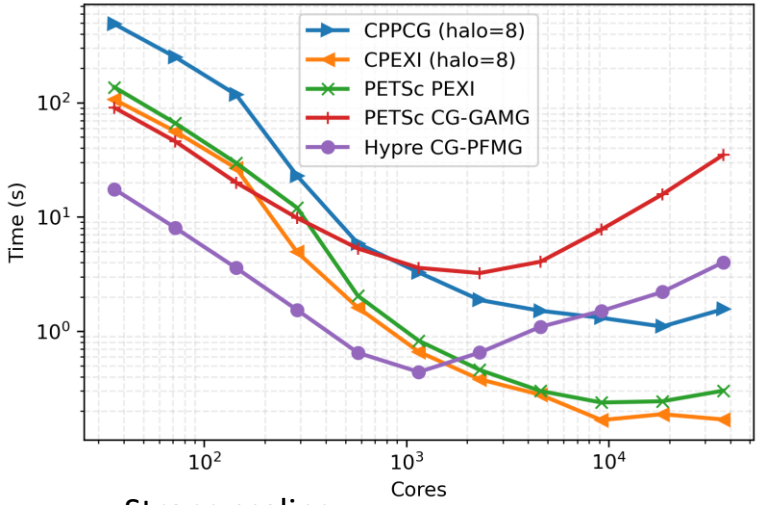


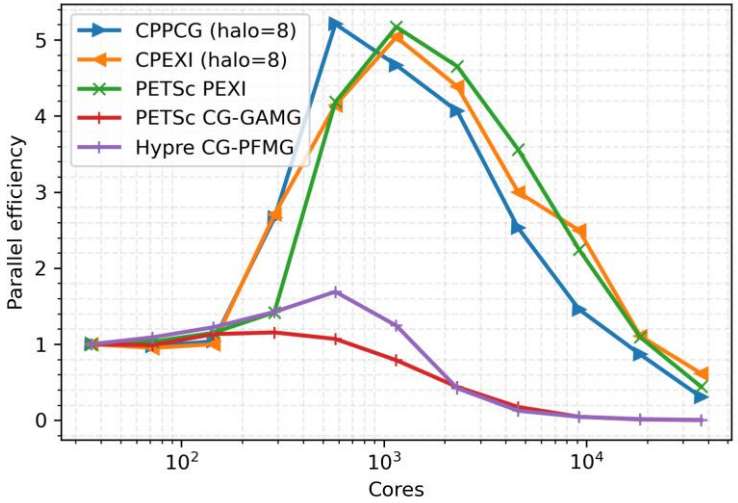






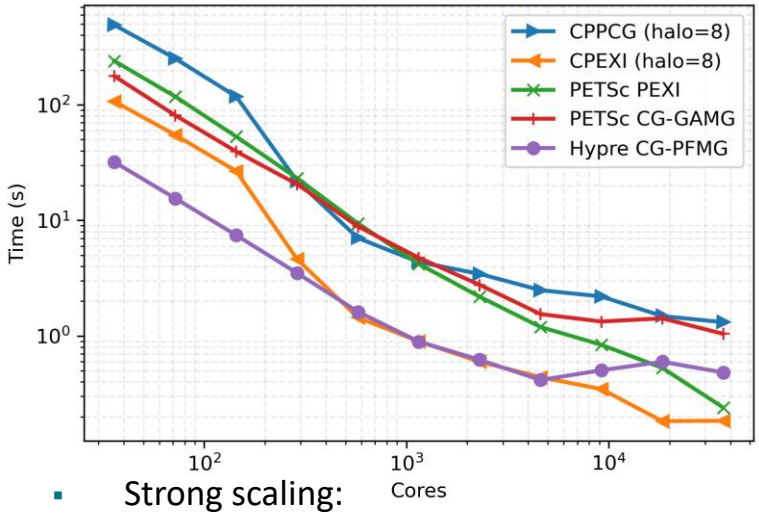
Strong scaling and efficiency (MPI x OMP = 36 x 1)

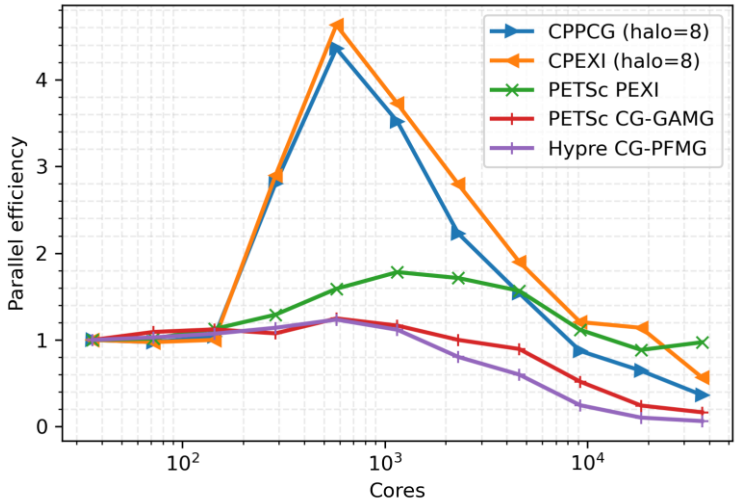




- Strong scaling:
  - Improved run time and scalability over previous CPPCG method.
  - Scalability plateaus at around 10000 cores.
- Parallel efficiency
  - Slightly improved compared CPPCG method at higher core counts.

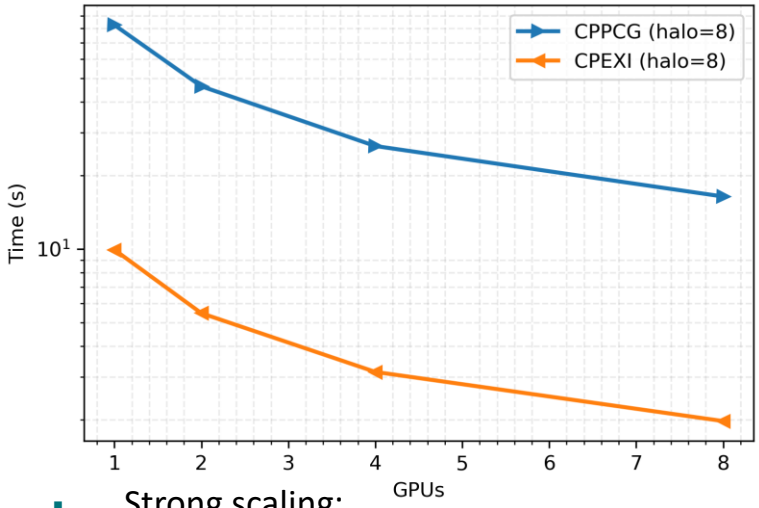
Strong scaling and efficiency (MPI x OMP = 6 x 6)

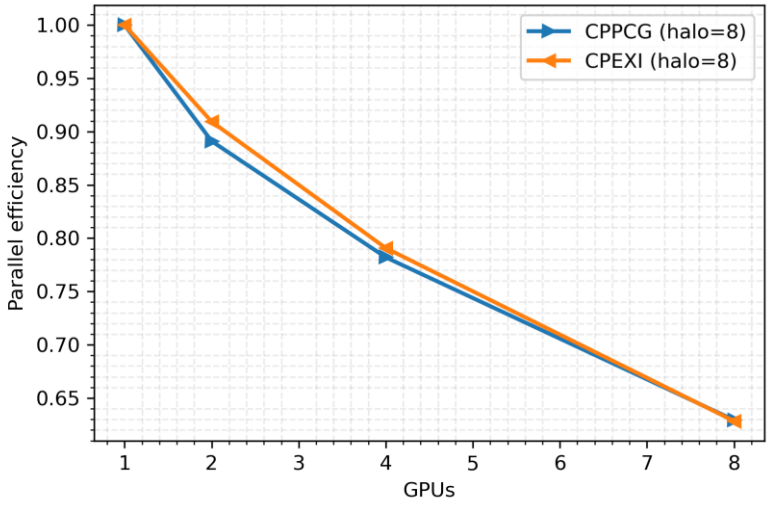




- Strong scaling:
  - Threading improves scalability of all methods.
  - Again, plateaus at around 10000 cores.
- Parallel efficiency
  - Similar to message of previous slide.

Strong scaling and efficiency (MPI+CUDA)





- Strong scaling:
  - Improved run time compared to CPPCG method.
  - Would expect to strong scale further with more GPUs
- Parallel efficiency
  - Similar efficiency to the CPPCG method.

Summary

- Scalable ETD implementation (CPEXI) within TeaLeaf mini-app.
- Discussed MPK used to reduce communications at scale.
- Demonstrated strong scalability and parallel efficiency for CPEXI.
- CPEXI outperforms previous CPPCG method in TeaLeaf mini-app.
- Competitive compared to best structured combination in Hypre (CG + PFMG).

- **Thank you. Questions?**

Back up slides

Hardware used

	Damson	Bullace
Nodes	3162	50 (4 with GPUs)
Interconnect	Infiniband EDR	Infiniband EDR
Proc type	Intel Xeon	Intel Xeon
Proc model	E5-2697v4	E5-2697v4
Clock (GHz)	2.3	2.3
GPU model	None	Nvidia V100
GPUs per node	0	2
Memory Bandwidth	71.53 GB/s	71.53 GB/s