Multicore Bundle Adjustment - Supplemental Material

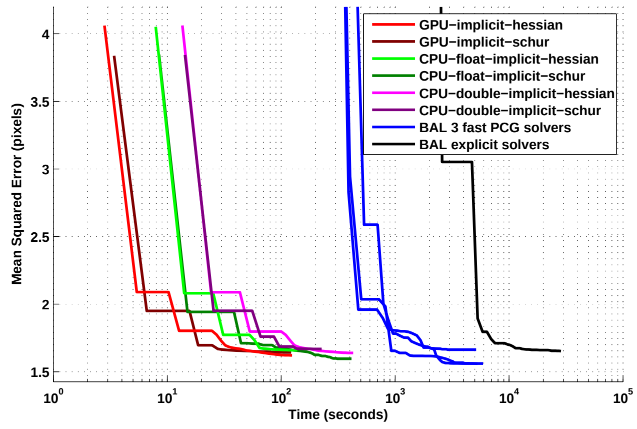
This document reports the comparison of runtime and convergence on various datasets.

- We selected 6 models (to cover different sizes) from each of the 5 datasets provided by Agarwal et al. [1]
  - Final .... results in Figure 1;
  - Venice Skeletal .... results in Figure 2;
  - Ladybug .... results in Figure 3.
  - Dubrovnik Skeletal ... results in Figure 4;
  - Trafalgar Skeletal ..... results in Figure 5;
- We compare the speed and convergence of our system with 5 algorithms of Agarwal et al. [1].
  - *explicit-sparse* which computes a sparse factorization of the Schur complement S
  - *explicit-jacobi* which explicit computes S and runs PCG on it using a block-Jaocbi precondoitioner
  - *normal-jacobi* which runs PCG on the H_λ with M_λ as preconditioner
  - *implicit-jacobi* and *implicit-ssor*, which implicitly apply the Schur complement trick by reusing the H_λ

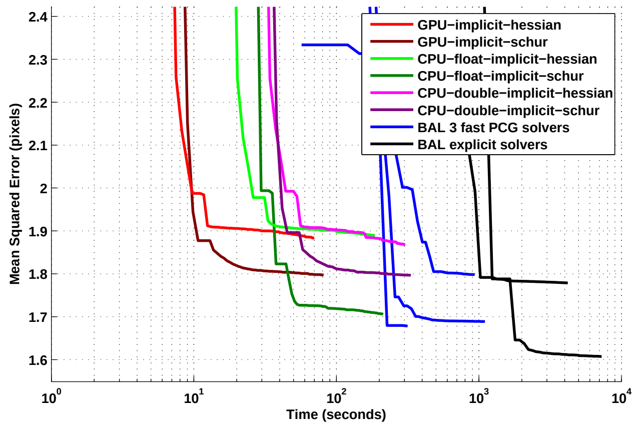
  For clear visualization of our algorithms, we divide the above algorithms into two groups:
  - *BAL 3 fast PCG solvers*: *normal-jacobi*, *implicit-jacobi* and *implicit-ssor*
  - *BAL explicit Solvers*: *explicit-sparse* and *explicit-jacobi*
- We run our algorithms in the following 6 modes (with the same single-precision input)
  - *GPU implicit-hessian*
  - *GPU implicit-schur*
  - *CPU float implicit-hessian* (multi-threaded single-precision)
  - *CPU float Implicit-schur* (multi-threaded single-precision)
  - *CPU double implicit-hessian* (multi-threaded double-precision)
  - *CPU double implicit-schur* (multi-threaded double-precision)
- The experiments show that our single-precision solvers and our double-precision solvers have the same convergence, which further validates our choice of single-precision.

References

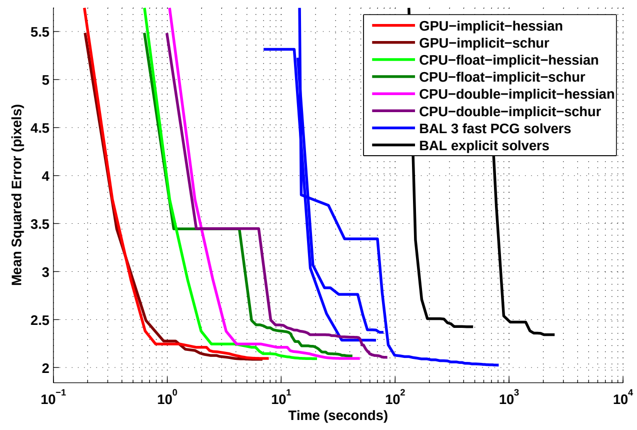
[1] S. Agarwal, N. Snavely, S. Seitz, and R. Szeliski. "Bundle Adjustment in the Large." In ECCV10, pages II: 29-42, 2010. http://grail.cs.washington.edu/projects/bal/ 1



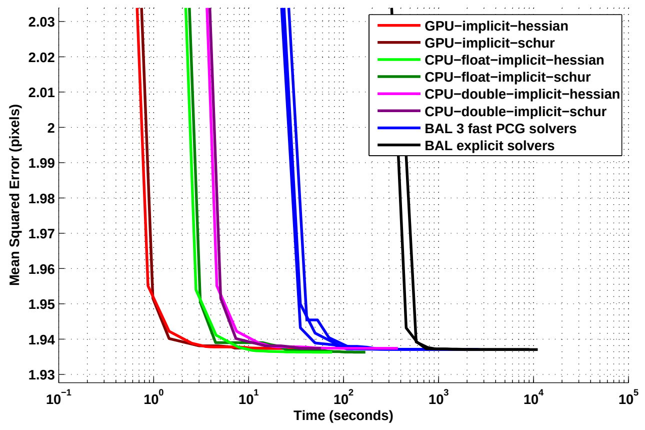
(a) $m = 13775; n = 4468275; k = 29823581$ (Venice)



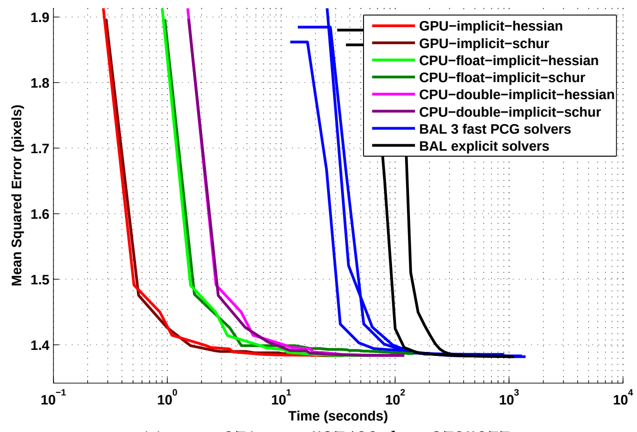
(b) $m = 4585; n = 1324582; k = 9125125$ (Dubrovnik)



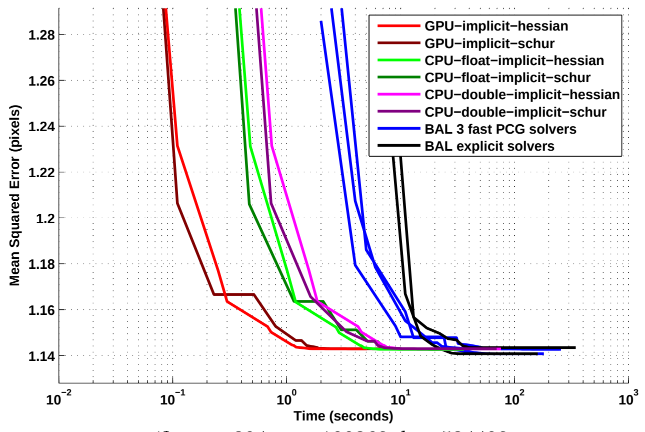
(c) $m = 3068; n = 310854; k = 1653812$ (Rome)



(d) $m = 1936; n = 649673; k = 5213733$ (Trafalgar)

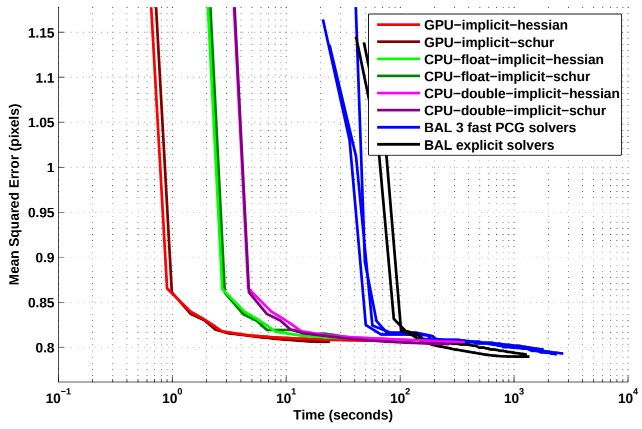


(e) $m = 871; n = 527480; k = 2785977$

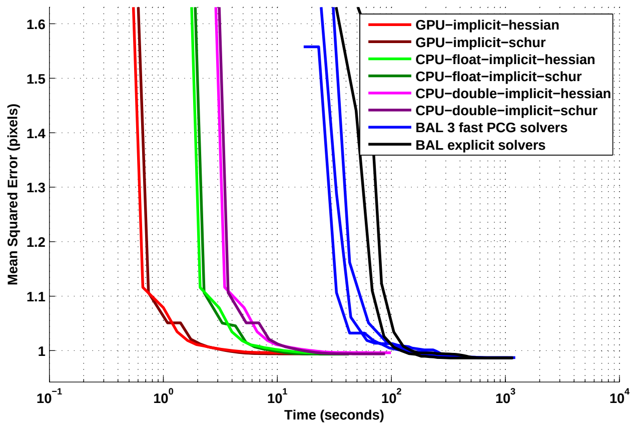


(f) $m = 394; n = 100368; k = 534408$

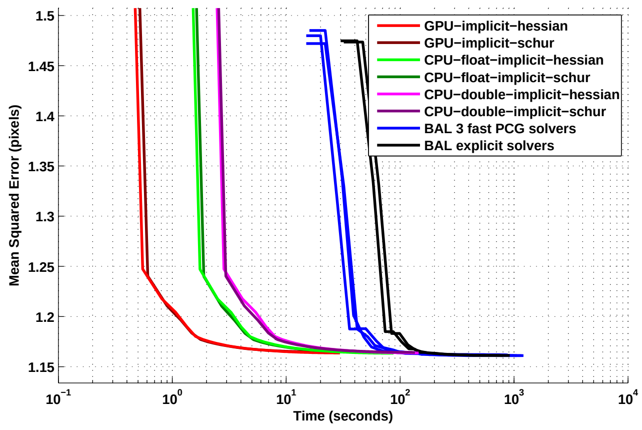
Figure 1. Runtime for 6 Final models.



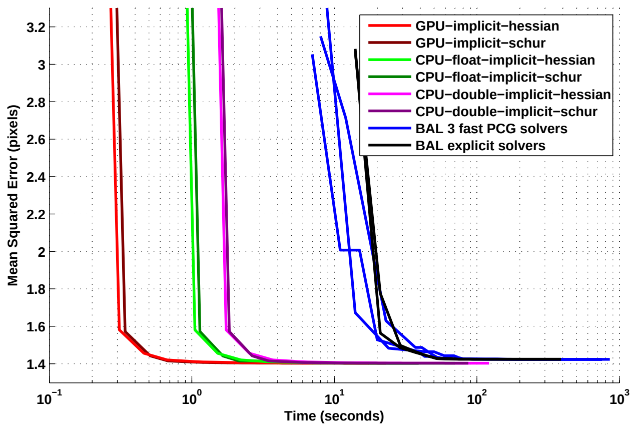
(a) $m = 1778; n = 993923; k = 5001946$



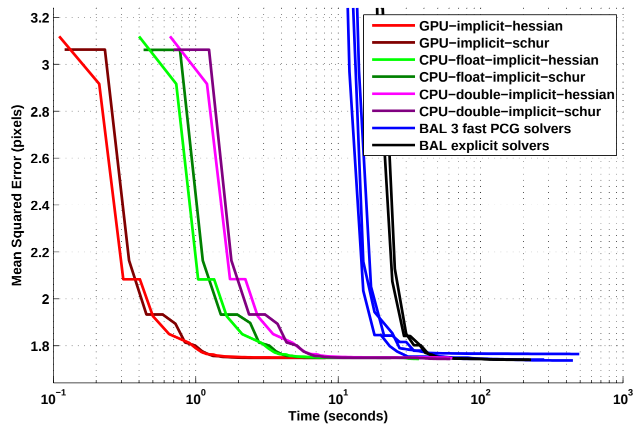
(b) $m = 951; n = 708276; k = 3748892$



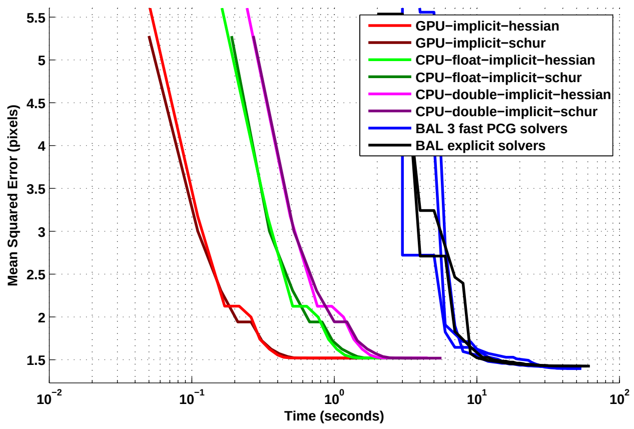
(c) $m = 744; n = 543562; k = 3058863$



(d) $m = 427; n = 310384; k = 1699145$

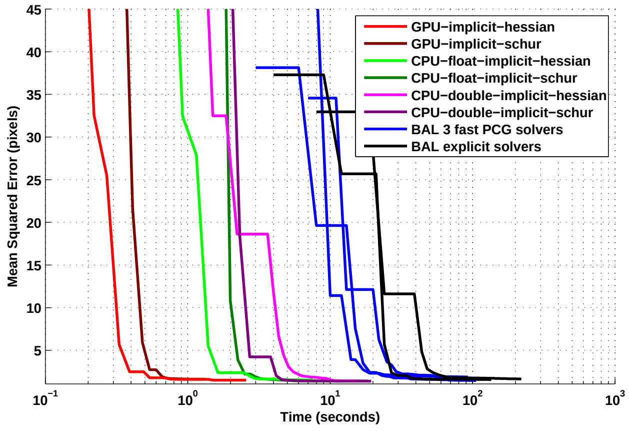


(e) $m = 245; n = 198739; k = 1091386$

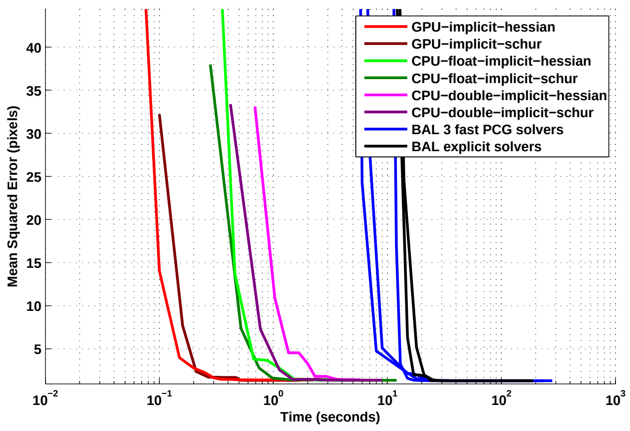


(f) $m = 52; n = 64053; k = 347173$

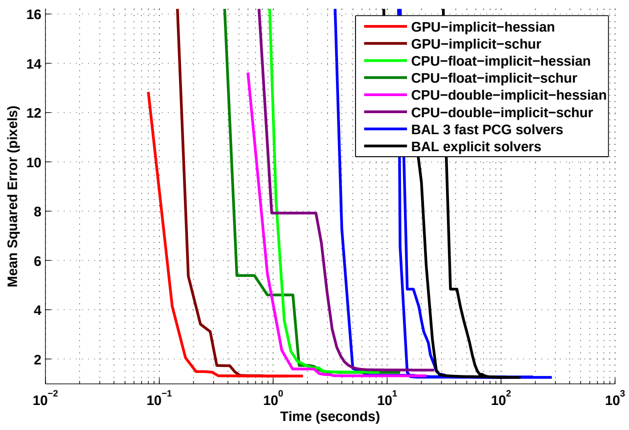
Figure 2. Runtime for 6 Venice Skeletal models.



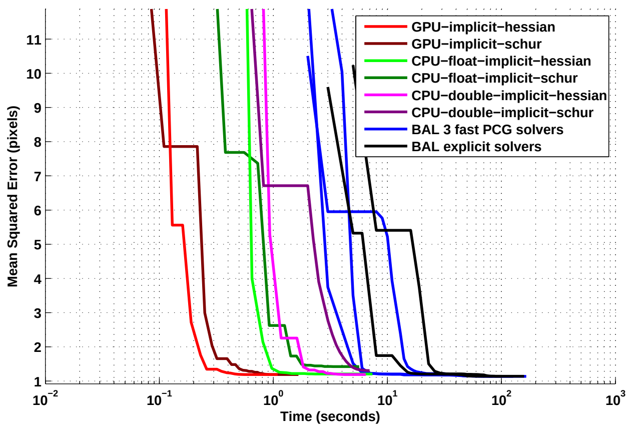
(a) $m = 1723; n = 156502; k = 678718$



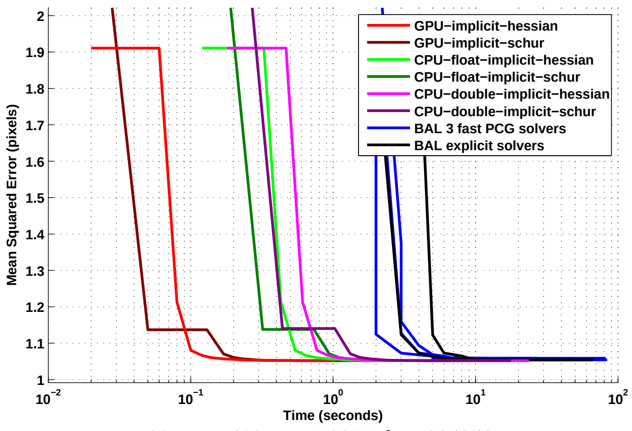
(b) $m = 1266; n = 132593; k = 587942$



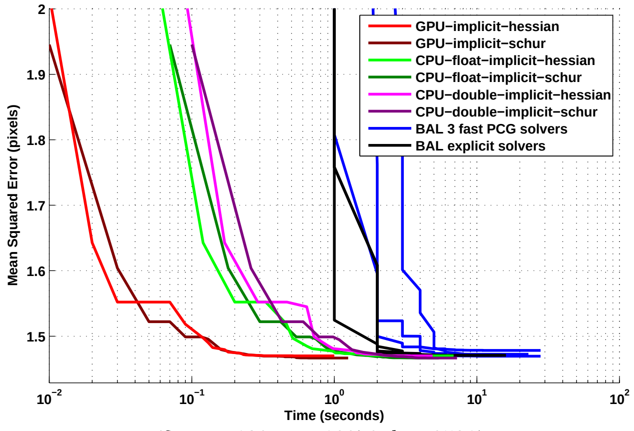
(c) $m = 1064; n = 113655; k = 510211$



(d) $m = 810; n = 88814; k = 393775$

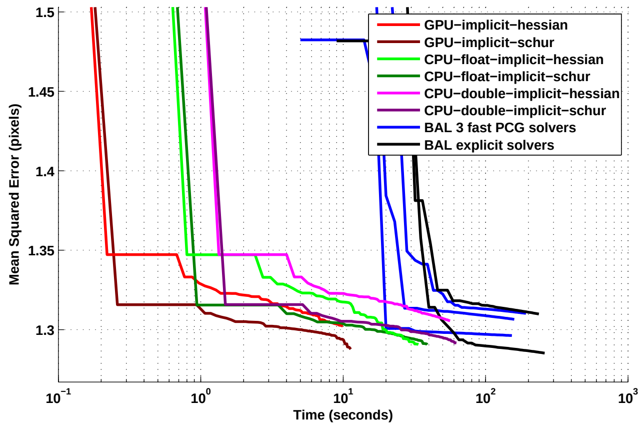


(e) $m = 412; n = 52215; k = 224242$

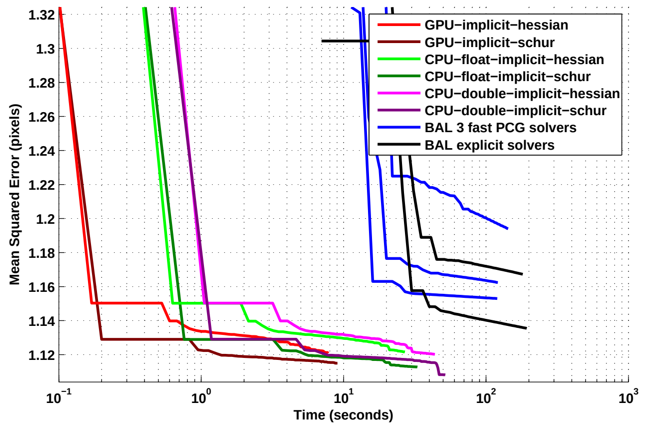


(f) $m = 138; n = 19878; k = 85217$

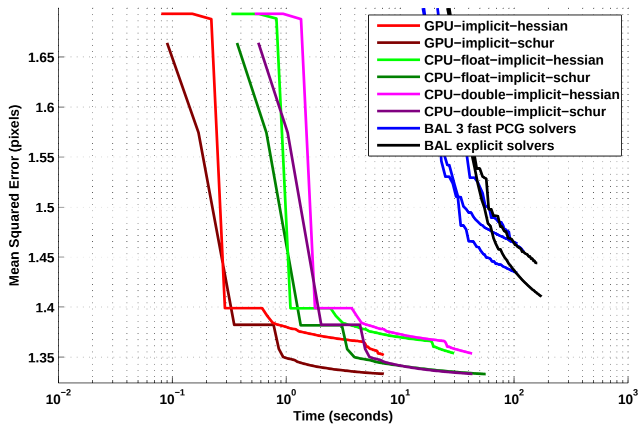
Figure 3. Runtime for 6 Ladybug models. Since this dataset is essentially a collection of video sequences, the Schur complements of them are relatively sparse.



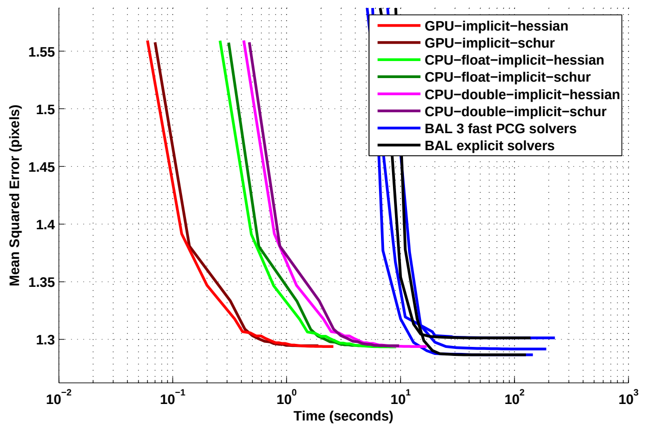
(a) $m = 356; n = 226730; k = 1255268$



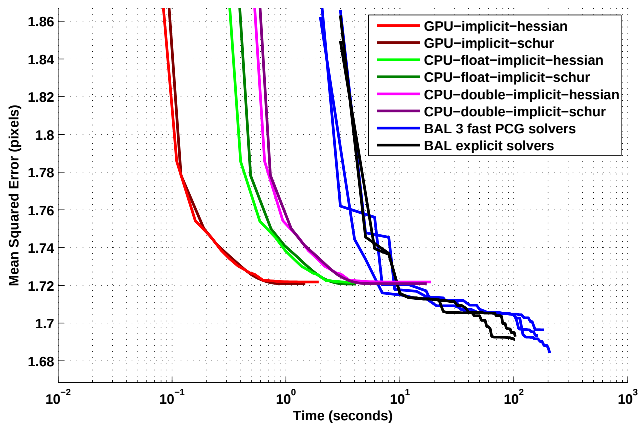
(b) $m = 273; n = 176305; k = 942970$



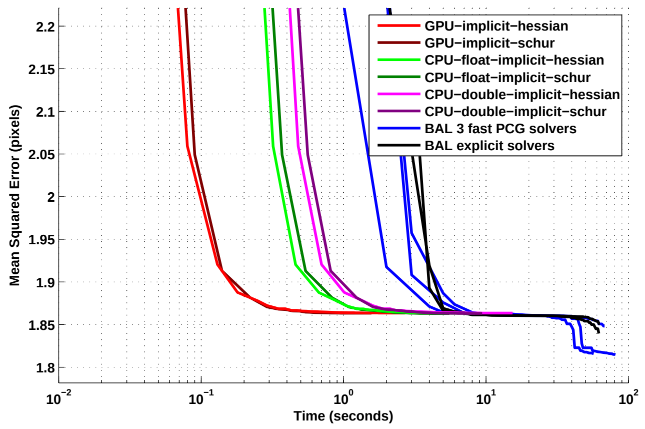
(c) $m = 237; n = 154414; k = 858331$



(d) $m = 182; n = 116770; k = 668705$

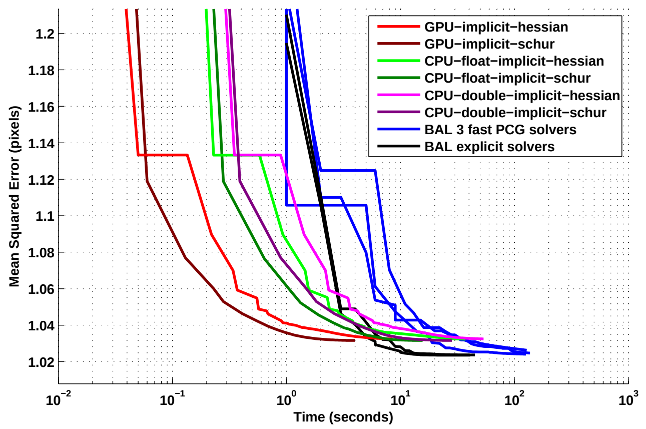


(e) $m = 135; n = 90642; k = 553336$

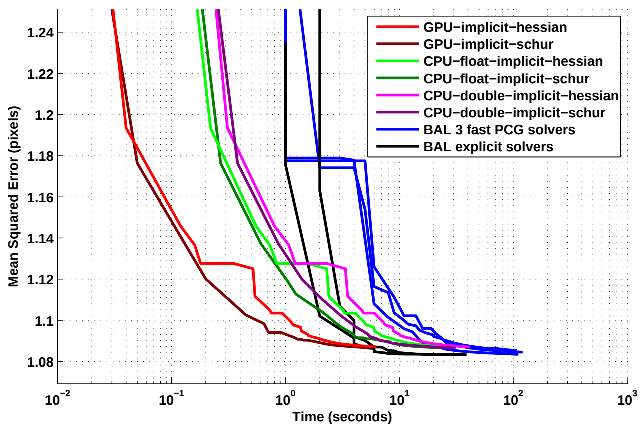


(f) $m = 88; n = 64298; k = 383937$

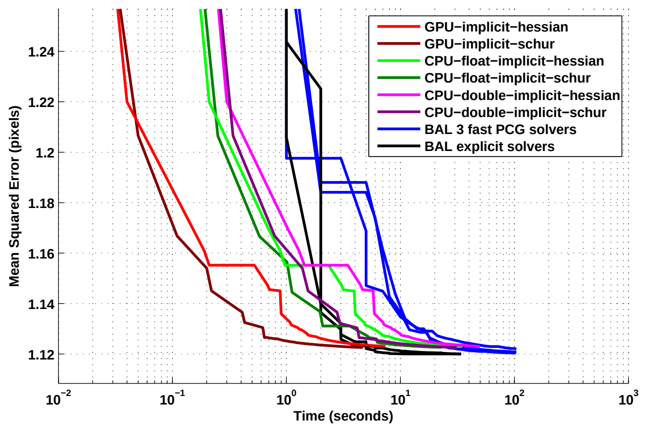
Figure 4. Runtime for 6 Dubrovnik Skeletal models.



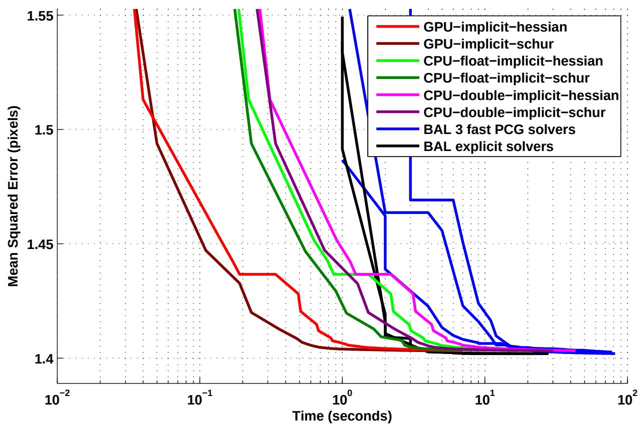
(a) $m = 257; n = 65132; k = 225911$



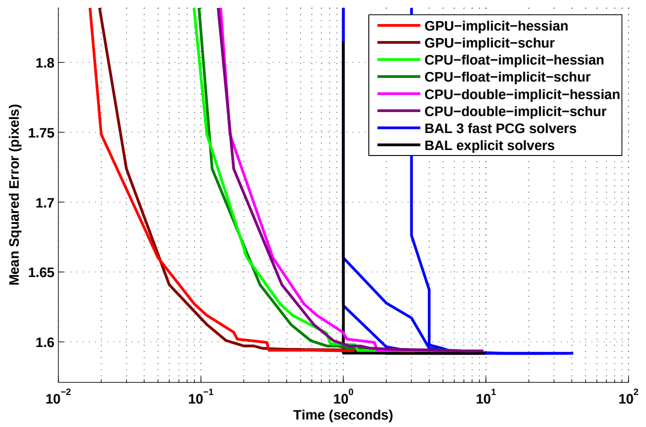
(b) $m = 206; n = 54562; k = 200715$



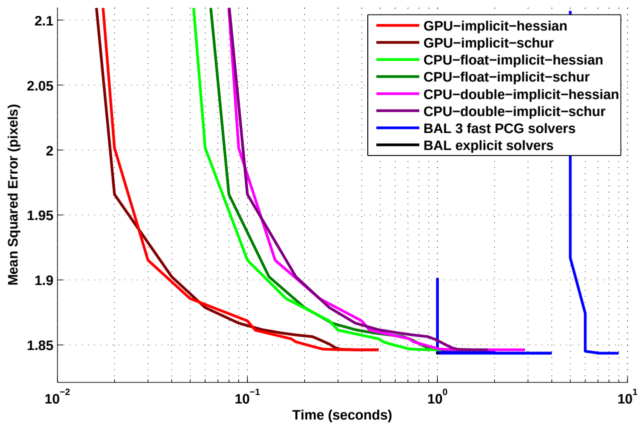
(c) $m = 170; n = 49267; k = 185815$



(d) $m = 126; n = 40037; k = 148328$



(e) $m = 50; n = 20431; k = 73967$



(f) $m = 21; n = 11315; k = 36455$

Figure 5. Runtime for 6 Trafalgar Skeletal models. This models of this dataset are very small. The almost vertical lines of BAL solvers in (e) and (f) are due to the low precision timing of the BAL code.