Abstraction Layer For Standardizing APIs of Task-Based Engines

الفرسان

R. Alomairy[+], H. Ltaief[+], M. Abduljabbar[*], and D. Keyes[+]

Extreme Computing Research Center, KAUST[+]
Chalmers University of Technology[*]

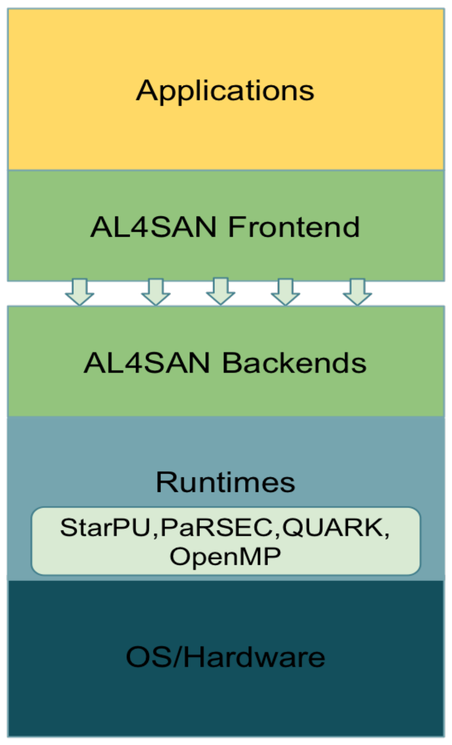
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology
Extreme Computing Research Center

AL4SAN

Motivations

AL4SAN is a lightweight library for abstracting the APIs of task-based runtime engines. AL4SAN unifies the expression of tasks and their data dependencies. It supports various dynamic runtime systems relying on compiler technology and user-defined APIs. It enables a single application to employ different runtimes and their respective scheduling components, while providing user-obliviousness to the underlying hardware configurations. AL4SAN exposes common front-end APIs and connects to different back-end runtimes. AL4SAN enables runtime interoperability by switching runtimes at runtime. Blending runtime systems permits to achieve a twofold speedup on a task-based generalized symmetric eigenvalue solver, relative to state-of-the-art implementations. The ultimate goal of AL4SAN is not to create a new runtime, but to strengthen co-design of existing runtimes/applications, while facilitating user productivity and code portability.

Stacked Structure



Runtimes and Features Accessible thru AL4SAN

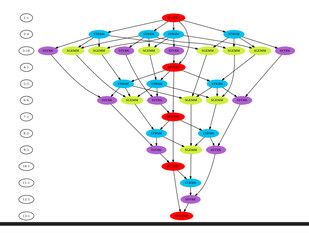
Runtimes	Shared Memory	Distributed Memory	GPU	Runtime Features	Task Model	Interface Languages	Compiler Support
OpenMP 4.0	OpenMP threads	✗	✗	GCC ICC CLANG Directive	Sequential task flow fork-join model	C C++ Fortran	✓
StarPU	Pthreads	✓	✓	prio, dm, dmda, eager work stealing	Sequential task flow	C	✗
QUARK	Pthreads	✗	✗	priority locality accumulator gatherv	Sequential task flow	C	✗
PaRSEC-DTD	Pthreads	✓	✓	LFQ/PBQ work stealing	Sequential task flow	C	✗

Sequential Task API

```
AL4SAN_Task_CPU(POTRF, POTRF_CPU)
#define A(m, n) AL4SAN_ADDR(A, double, m, n)
void potrf_task(AL4SAN_option_t * options, char uplo,
  int nb, AL4SAN_desc_t *A, int m, int n, int lda, int info)
  {
    AL4SAN_Insert_Task(AL4SAN_TASK(POTRF),
      options,
      AL4SAN_VALUE,   &uplo,    sizeof(int),
      AL4SAN_VALUE,   &nb,      sizeof(int),
      AL4SAN_INOUT,   A (m, n), AL4SAN_DEP,
      AL4SAN_VALUE,   &lda,     sizeof(int),
      AL4SAN_VALUE,   &info,    sizeof(int),
      ARG_END);}
void POTRF_CPU (AL4SAN_arglist *al4san_arg) {
    char uplo;
    AL4SAN_Unpack_Arg(al4san_arg, &uplo, &nb, &A,
  &lda, &info );
    potrf_(&uplo, &nb, A, &lda, &info); }
```

Dense Cholesky Example

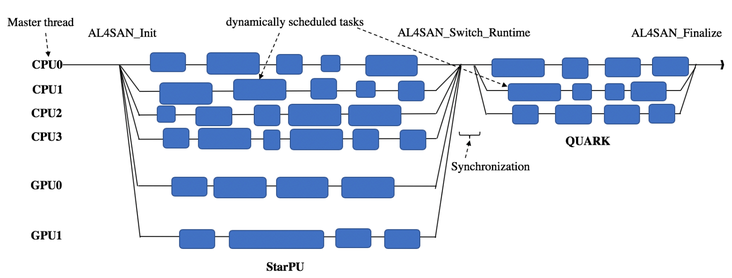
```
AL4SAN_Init(runtime, ncpus, ngpus);
request = REQUEST_INITIALIZER;
sequence=AL4SAN_Sequence_Create();
options= AL4SAN_Options_Init(sequence, request);
AL4SAN_Matrix_Create(&A, NULL, sizeof(double),
  AL4SAN_Col_Major, nb, nb, bld, rows, cols, ld);
for k ← 0 to nt do
    potrf_task(options, 'L', nb, A, k, k, lda, info);
    for m ← k + 1 to nt do
        trsm_task(options, 'R', 'L', 'T', 'N', A, k, k, lda, A,
          m, k, ldb);
    for n ← k + 1 to nt do
        syrk_task(options, 'L', 'N', nb, nb, -1.0, A, n, k,
          lda, 1.0, A, n, n, ldb);
        for m ← n + 1 to nt do
            gemm_task(options, 'N', 'T', nb, nb, nb, -1.0,
              A, n, k, lda, A, m, k, ldb, 1.0, A, n, m, ldc);
AL4SAN_Options_Finalize(options);
AL4SAN_Sequence_Wait( sequence);
AL4SAN_Sequence_Destroy(sequence);
AL4SAN_Finalize();
```



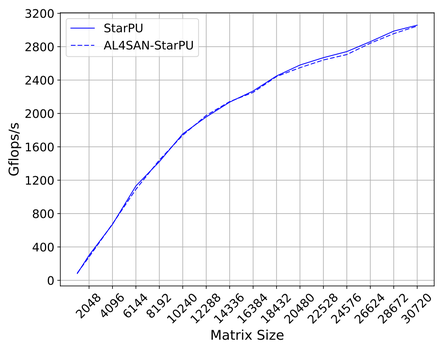
Objectives

- Standardizing task-based runtime systems
- Relying on a lightweight abstraction layer
- Supporting different hardware architectures
- Improving user productivity
- Leveraging runtime interoperability
- Achieving limited overhead (up to 10%)

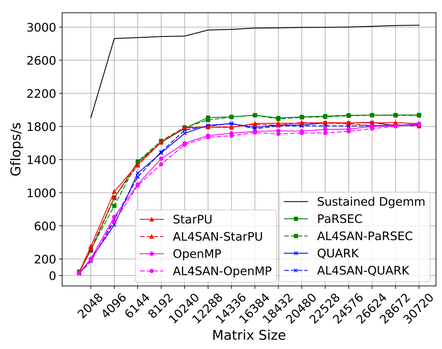
Runtime Interoperability: StarPU - QUARK



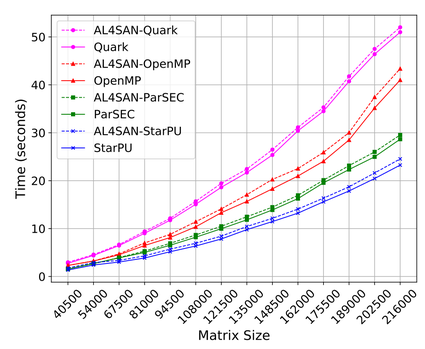
Performance Results



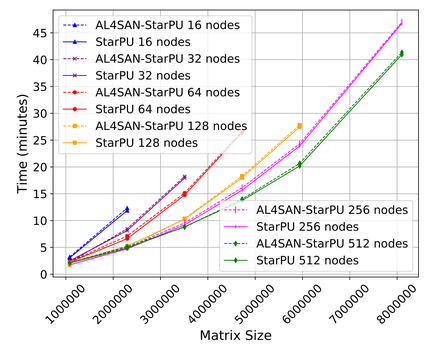
Dense Cholesky on Dual-Socket
14-core Intel Broadwell with 8x Nvidia K80s



Dense Cholesky on Dual-Socket
28-core Intel Skylake

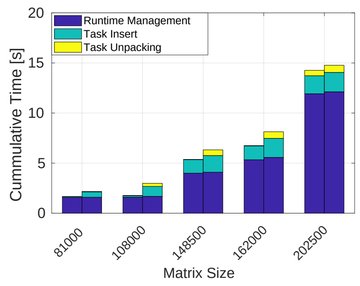


Tile Low-Rank Cholesky on Dual-Socket
28-core Intel Skylake

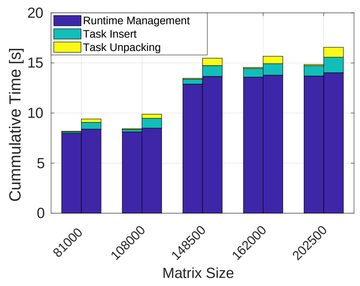


Tile-Low Rank Cholesky on Cray XC40
with Dual-Socket 16-core Intel Haswell System

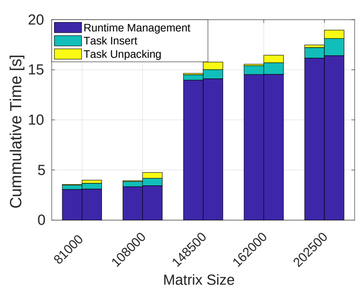
AL4SAN overhead is small (Skylake example)



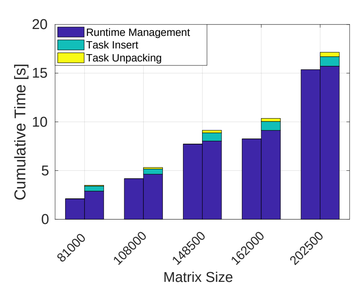
Left: PaRSEC, right: AL4SAN-PaRSEC



Left: QUARK, right: AL4SAN-QUARK

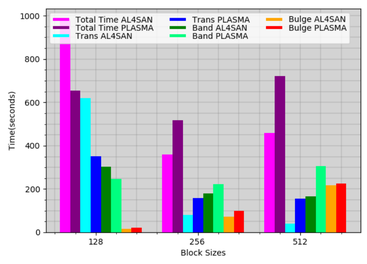


Left: StarPU, right: AL4SAN-StarPU

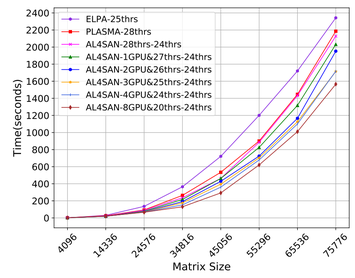


Left: OpenMP, right: AL4SAN-OpenMP

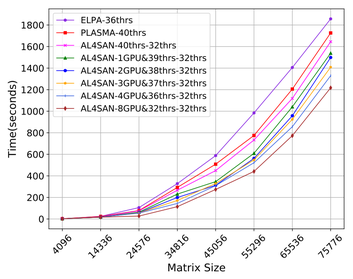
AL4SAN's multi-runtime wins on Generalized Symmetric Eigensolver



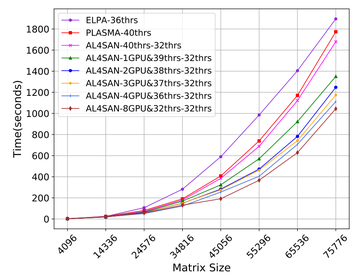
Dual-Socket 14-core Intel Broadwell with 8x Nvidia K80s



Dual-Socket 14-core Intel Broadwell with 8x Nvidia K80s



Dual-Socket 20-core Intel Broadwell with 8x Nvidia P100s



Dual-Socket 20-core Intel Broadwell with 8x Nvidia V100s

Future Roadmap

- Extending support to more engines
- Leveraging data abstraction
- Integrating C++ constructs
- Showcasing with more algorithms and applications

Main Reference and Software Release

AL4SAN, **2020**, R. Alomairy, H. Ltaief, M. Abduljabbar, and D. Keyes, IEEE Trans. Par. Dist. Sys. doi:10.1109/TPDS.2020.2992923.

Acknowledgment

- INRIA/INP/LaBRI Bordeaux
- ICL @ UTK
- KSL @ KAUST
- NVIDIA
- Intel
- Cray/HPE