Supplemental Material: Thallo – Scheduling for High-Performance Large-scale Non-linear Least-Squares Solvers

MICHAEL MARA, FELIX HEIDE, MICHAEL ZOLLHÖFER, PAT HANRAHAN, MATTHIAS NIESSNER

ACM Reference Format:
Michael Mara, Felix Heide, Michael Zollhöfer, Pat Hanrahan, Matthias Nießner. 2021. Supplemental Material: Thallo – Scheduling for High-Performance Large-scale Non-linear Least-Squares Solvers. *ACM Trans. Graph.* 1, 1, Article 1 (January 2021), 11 pages. https://doi.org/10.1145/3453986

This supplemental document is divided into the following sections:

- Section 1 is an expanded description of the various applications we implemented in Thallo.
- Section 2 gives absolute timing information for the architectures in the exhaustive autoscheduling experiment.
- Section 3 provides a more detailed description of the DSL itself.
- Section 4 provides a brief discussion and analysis of code complexity for applications written in our system.
- Section 5 is a brief validation of solvers generated in Thallo on the classic NIST StRD benchmark.
- Section 6 is a more in-depth evaluation of the Bundle Adjustment example, which is a heavily studied application.
- Section 7 provides an evaluation of the predictiveness of the cost model used in the autoscheduler.

1 APPLICATIONS IN DEPTH

We provide slightly expanded descriptions of each of the applications used in the paper here. For easy reference, we present the timing results here again in Table 1.

1.1 BundleFusion (Sparse and Dense)

Bundle Adjustment is at the core of every structure-from-motion framework (SfM). It is used estimate accurate and globally-consistent camera parameters alongside a sparse 3D reconstruction [Agarwal et al. 2011; Jebara et al. 1999; Schönberger and Frahm 2016; Triggs et al. 2000] from a set RGB images. BundleFusion [Dai et al. 2017] formulates the analog problem for the RGB-D case with known depth. It uses both sparse image correspondences and dense depth maps to achieve highly-accurate loop closure even for large-scale indoor scenes based on a hand-written data-parallel GPU solver. Figure 1 shows an example of scene reconstruction before and after optimization using a solver generated by Thallo.

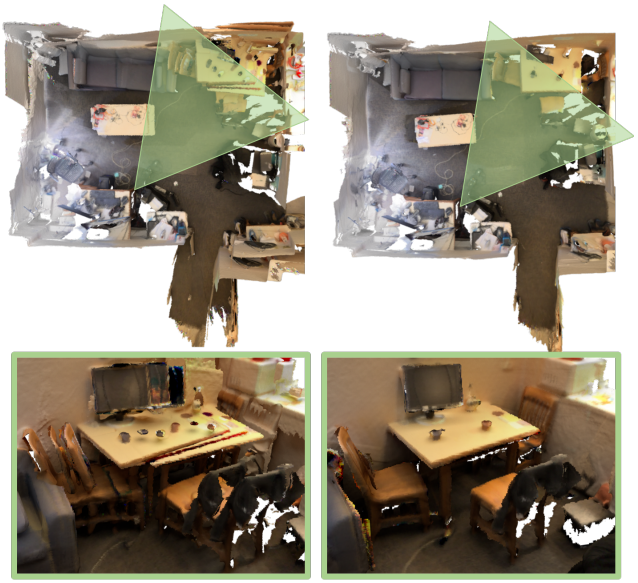

Fig. 1. An example of 3D reconstruction before and after optimizing the camera poses with solvers generated by Thallo. Global optimization increases the global consistency of camera poses and thus improves reconstruction quality.

For a sparse set of inter-frame correspondences C and a set of N images, a global alignment energy is optimized to find the best rigid camera transformations $\mathbf{T}_i$ such that the total point-to-point error of a sparse set of detected and matched feature points is minimized:

$$\mathbf{T}_i^* = \arg\min_{\mathbf{T}_i} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{(l,k)\in C(i,j)} \left\| \mathbf{T}_i \mathbf{c}_i^k - \mathbf{T}_j \mathbf{c}_j^l \right\|_2^2 . \quad (1)$$

Here, $C(i,j)$ is the set of all correspondences (k,l) between the i-th and j-th frame and $\mathbf{c}_i^k$ is the 3D position of the k-th detected feature point in frame i. The rigid transformations $\mathbf{T}_i \in \mathbb{R}^{4\times4}$ encode camera rotation and translation and map the 3D points from camera to world space.

In addition to the sparse alignment term of BundleFusion [Dai et al. 2017], we demonstrate the optimization of the dense geometric alignment terms using Thallo, on a representative 11-frame chunk, which is the most challenging real-time optimization problem tackled in BundleFusion. This increases the overall number of residuals from about 2000 (for the sparse term only) to > 265000. In addition, the complexity of the scheduling drastically increases since the exact number of dense residual terms depends on the available depth values as well as found matches for each frame, which makes implementing a hand-crafted implementation particularly

Author's address: Michael Mara, Felix Heide, Michael Zollhöfer, Pat Hanrahan, Matthias Nießner.

0730-0301/2021/1-ART1 $15.00
https://doi.org/10.1145/3453986

	Optimization Problem						
Solver	BundleFusion [2017]	Face Fit [2016]	Shape/Shading	Bundle Adjust [2000]	Deconv. (11 × 11)	Deconv. (15 × 15)	SV Deconv.
Ceres [2010a]	5394.10ms	25612.1ms	21982.3ms	11954.56ms	25304.85ms	55158.87ms	81344.37ms
Opt DSL [2017]	110.39ms	1057.8ms	2259.6ms	114.78ms	195.93ms	∞	∞
Hand-written CUDA	5.57ms	1946.5ms	N/A	N/A	N/A	N/A	N/A
Ours (Unscheduled)	106.3ms	1013.1ms	2235.6ms	105.81ms	187.73ms	253.39ms	452.16ms
Ours (Auto)	**2.96ms**	**46.3ms**	**171.86ms**	**89.47ms**	**46.58ms**	**93.11ms**	**137.92ms**

Table 1. Copy of main result table from the paper: time-to-convergence of different solvers on various applications. From top to bottom, we have solvers generated by the Ceres library [Agarwal et al. 2010a], solvers generated by Opt [Devito et al. 2017], handwritten expert CUDA solvers (using hand-coded derivatives) if existent, solvers generated by Thallo with no scheduling annotations (and no autoscheduling), and solvers generated by Thallo with autoscheduling. The fastest implementation for each problem is bolded. Note that Ceres is a flexible high-level library that executes on the CPU and thus executes on different (lower-powered) hardware, while the other systems use the more powerful GPU. Also, Opt fails to compile for 15 × 15 Deconvolution and Spatially-Varying Deconvolution.

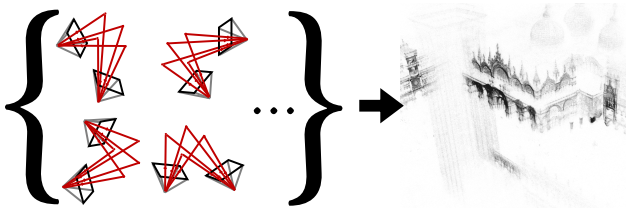

Fig. 2. Bundle Adjustment takes as input pairwise correspondences between 2D points in a set of images and solves for globally consistent camera positions, rotation, and distortion parameters, alongside 3D locations for the image points. This produces a sparse 3D map of the scene captured by the images.

challenging. Thallo outperforms the state-of-the-art hand-written GPU solver of [Dai et al. 2017] by over 30% and is over 35× faster than other high-level systems (Opt/Ceres); see Table 1.

1.2 Bundle Adjustment

For completeness, we implemented standard Bundle Adjustment, formulated as in Bundle Adjustment in the Large [Agarwal et al. 2010b]. In this section we evaluate the solvers on the second largest problem in the BAL dataset. The optimization is posed over 4585 cameras, each of which has 9 unknown parameters. There are 1324582 correspondences, and twice as many residual terms. We solve to an error threshold of $\tau = 0.1$. See Section 6 for a much more comprehensive comparison on all of the BAL problems, and with stricter error thresholds.

Thallo is over 100× faster than Ceres at this high error tolerance. It also takes 20% less time than the equivalent Opt solver, despite the Thallo schedule chosen strongly resembling the de facto Opt schedule; see Table 1.

1.3 Shape-and-Shading

Shape-From-Shading refines depth data from an RGB-D sensor using the detailed color image and a low-order spherical harmonic estimation of the lighting. The original realtime work in [Wu et al. 2014] and reimplemented in Opt [Devito et al. 2017] estimated the lighting in a preprocessing step and then fixed it while optimizing just for the depth values. We instead solve the significantly more complicated Shape-*and*-Shading problem by solving jointly for the depth values *and* the lighting conditions. This incorporates the nine spherical harmonic values to all of the shading residuals as additional unknowns. We also add an explicit lighting term that directly penalizes the difference between computed and input shading values (as opposed to just the gradients of those terms). Figure 3 shows an example of depth refinement before and after optimization using a solver generated by Thallo. This produces a significantly different Jacobian structure with far more nonzero terms than the original problem, and significant shared computation among different residuals and their partial derivatives. The autoscheduled solver is 13× faster than the default solver which materializes nothing outside of the inner loop.

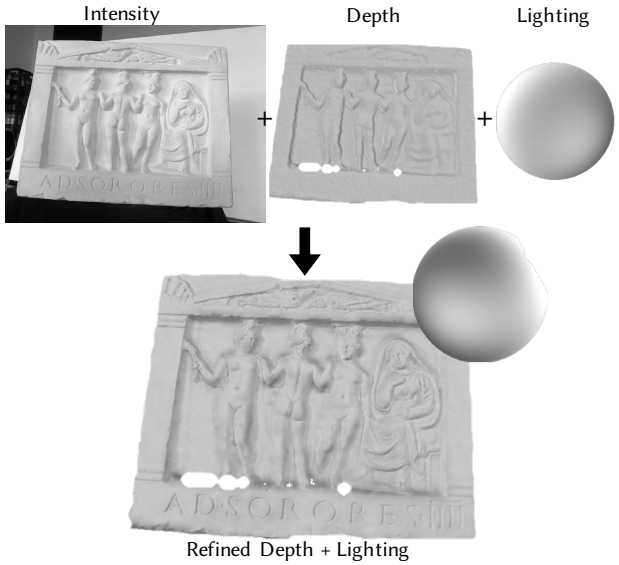

Fig. 3. An example of jointly refining depth data and an initial lighting estimate using a detailed color image for shading penalty terms. We refer to this optimization application as Shape-and-Shading, in contrast to Shape-from-Shading which keeps the lighting estimate fixed.

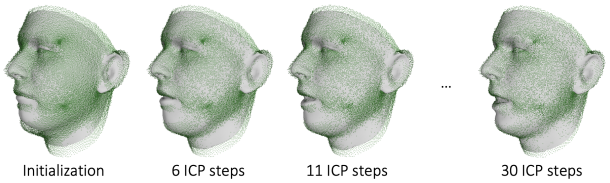

Fig. 4. An example of parametric model fitting using optimization.

1.4 Face Template Fitting

Non-rigid model-based registration is a fundamental building block for a large variety of tracking and reconstruction approaches [Loper et al. 2015; Romero et al. 2017; Thies et al. 2016; Xu et al. 2018]. One prominent example is fitting an affine parametric 3D mesh model $\mathcal{P}$ to 2D observations, e.g., as done in the Face2Face [Thies et al. 2016] approach. In the following, we assume that the embedding of a mesh is represented by a vector $\mathbf{m} \in \mathbb{R}^{3N}$ that stacks its N vertices. In the case of face tracking, the affine parametric model is then defined by a low-dimensional expression subspace of M blendshapes spanned by a matrix $\mathbf{B} \in \mathbb{R}^{3N \times M}$ that models the facial expressions relative to a neutral face template $\mathbf{a} \in \mathbb{R}^{3N}$:

$$\mathbf{m} = \mathcal{P}(\mathbf{c}) = \mathbf{a} + \mathbf{B}\mathbf{c} \quad .$$

Each of the M blendshapes represents a different facial expression as per-vertex offsets to the neutral face template. New faces $\mathbf{m} \in \mathbb{R}^{3N}$ are created by interpolation of these M displacement vectors based on the M blendshape weights $\mathbf{c} \in \mathbb{R}^{M}$. We fit the model to 2D data by finding the best coefficients $\mathbf{c}^*$, such that $\Theta(\mathcal{P}(\mathbf{c}^*))$ best matches a set of target correspondences $\mathbf{t} \in \mathbb{R}^{2N}$, where Θ is a projective camera transformation (using the 9-dimensional parameterization from BAL [Agarwal et al. 2010b]):

$$\mathbf{c}^* = \arg\min_{\mathbf{c}} \left\| \mathbf{t} - \Theta(\mathcal{P}(\mathbf{c})) \right\|_2^2 \quad . \tag{2}$$

Typically, the number of observations is much larger than the number of blendshapes $N \gg M$, hence the resulting Jacobian matrix has a lot more rows than columns and the Jacobian matrix is fully dense. This optimization is run in alternation with a correspondence search to build a model-based non-rigid ICP. We generate solvers for this optimization with Thallo and compare to Opt and a hand written solver. Opt has no reduction construct, so summing along the $M = 140$ basis shapes requires an explicit loop in the energy, and an explicit connectivity structure that consumes $4n(2m + 1)$ bytes for M blendshapes with N vertices, far in excess of all other data required for the problem . In fact, the schedule used by Opt generates code that fails to compile at $M = 210$ as the per-residual code exceeds Cuda register limits. Using Thallo's `parallelize_sum` scheduling construct, we parallelizing along the rows as well as the columns of the Jacobian to materialize it and then do dense matrix-matrix and matrix-vector multiplies; this allows Thallo to outperform other high-level solvers; see Tab. 1.

1.5 Scheduling ProxImaL Optimization Problems

We evaluated a method for using Thallo on a general class of proximal optimization problems [Parikh and Boyd 2013] for imaging applications, scheduling the sub-problems of the ProxImaL optimization compiler [Heide et al. 2016]. By implementing efficient non-linear solves, our proximal backbone not only allows for significant runtime performance improvements but also enables non-linear problem instances that could previously not be tackled due to a lack of efficient proximal operators. The elevated class of image optimization problems follows the structure proposed in [Heide et al. 2016], a sum of penalties f_i on linear transforms $\mathbf{K}_i\mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^n$ as the unknowns:

$$\underset{\mathbf{x}}{\operatorname{argmin}} \sum_{i=1}^{I} f_i\left(\mathbf{K}_i\mathbf{x}\right) \quad \text{with} \quad \mathbf{K} = \begin{bmatrix} \mathbf{K}_1 \\ \vdots \\ \mathbf{K}_I \end{bmatrix}, \tag{3}$$

where $\mathbf{K} \in \mathbb{R}^{m \times n}$ is one large matrix that is composed of stacked linear operators $\mathbf{K}_1, \ldots, \mathbf{K}_I$. The linear operator $\mathbf{K}_i \in \mathbb{R}^{m_i \times n}$ selects a subset of m_i rows of $\mathbf{K}\mathbf{x}$. This subset of rows is then the input for the penalty functions $f_i : \mathbb{R}^{m_i} \rightarrow \mathbb{R}$. In existing methods, including [Figueiredo and Bioucas-Dias 2009, 2010], these penalty functions were restricted either to point-wise operators with efficient separable implementations, or the corresponding $\mathbf{K}$ as structured operator, e.g., spatially-invariant convolution kernels. In other words, these methods assume that an efficient proximal operator for f is present, which is formalized as

$$\mathbf{prox}_{\tau f}(\mathbf{v}) = \underset{\mathbf{x}}{\operatorname{argmin}} \left(f(\mathbf{x}) + \frac{1}{2\tau}\|\mathbf{x} - \mathbf{v}\|_2^2 \right),$$

with scalars $\tau > 0$ and $\mathbf{v} \in \mathbb{R}^{m_i}$, please see [Parikh and Boyd 2013] for a detailed description. The proposed method relaxes these restrictions and allows for non-local, non-linear least-squares functions, in addition to the separable ones.

We build on top of the ProxImaL compiler, that solves problems in the form of Problem (3) using choices of proximal algorithms, including the popular Chambolle-Pock [Chambolle and Pock 2011] and Half-Quadratic Splitting [Robini and Zhu 2015]. We modify splitting partitions Ω and Ψ of the set of functions $\{f_1, \ldots, f_I\}$ by including all least-squares, linear or non-linear in Ω, and all point-wise functions in Ψ. With this splitting, we define an example of the Alternating Direction Method of Multipliers [Boyd et al. 2001] in Algorithm 1. The algorithm runtime is dominated by the least-squares problem in the first update, as all other point-wise root-finding problems can be implemented either as analytic solutions or using efficient iterative methods [Parikh and Boyd 2013]. We replace the

ALGORITHM 1: ADMM to solve Problem (3)

Initialization: $\rho > 0$, $\alpha \in (0, 2)$, $(\mathbf{x}^0, \mathbf{z}^0, \lambda^0)$
for $k = 1$ *to* V **do**
$\mathbf{x}^{k+1} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \sum_{i \in \Omega} f_i(\mathbf{x}) + \sum_{j \in \Psi} (\rho/2)\|\mathbf{K}_j\mathbf{x} - \mathbf{z}_j^k + \lambda_j^k\|_2^2$
$\mathbf{z}_j^{k+1} = \mathbf{prox}_{\frac{f_j}{\rho}}\left(\mathbf{K}_j(\alpha\mathbf{x}_j^{k+1} + (1-\alpha)\mathbf{x}_j^k) + \lambda_j^k\right) \quad \forall j \in \Psi$
$\lambda_j^{k+1} = \lambda_j^k + \mathbf{K}_j\left(\alpha\mathbf{x}_j^{k+1} + (1-\alpha)\mathbf{x}_j^k\right) - \mathbf{z}_j^{k+1} \quad \forall j \in \Psi$
end

CG and LSQR implementations for this dominating sub-problem, which rely on unscheduled matrix-free function calls, with solvers generated by Thallo. We can also fold non-linear functions into this

step which cuts off nested splitting and leads to improved convergence. Our Thallo-ProxImaL bridge handles expensive least-square problems, linear or non-linear, while keeping the existing Halide kernels for straight-forward point-wise operations and matrix-vector product evaluations.

1.5.1 Poisson Deconvolution. Fig. 5 shows an example of Poisson-deconvolution for Nexus 5 smartphone optical blur. Although modern smartphone optics feature highly-optimized stacks of up to six individual lens elements, residual aberrations may still occur due to manufacturing tolerances or imperfections in the individual lens elements. We calibrated these aberrations for a Nexus 5 rear camera using the method from Mosleh et al. [2015]. The spatially varying aberrations are shown on the right. While existing methods require these aberrations to be local point-spread-functions (PSFs), i.e., spatially-invariant stencil-operations in a local area, Thallo lets us formulate per-pixel PSFs with arbitrary support over the image plane. We adopt the Poisson optimization problem from [Heide et al. 2016] with identical regularization weights on the total-variation

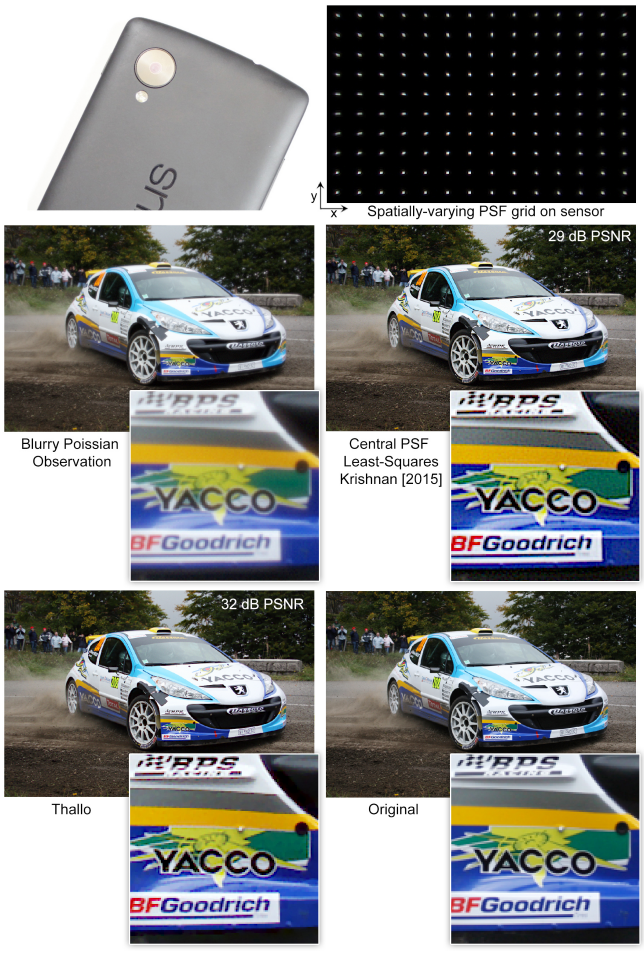

Fig. 5. Spatially-varying deconvolution. This problem cannot be expressed in the original ProxImaL framework, but can with the new Thallo backend.

term and schedule all quadratic problems with our efficient Thallo solver. Compared to methods that assume Gaussian noise and a single convolutional operator, such as the popular approach of Krishnan et al. [2011], the proposed model allows for accurate reconstruction results, reducing residual artifacts and ringing around fine details. Note that we improve on the baseline because Thallo allows us to more accurately model dense image formation models.

Not only does Thallo improve image quality, but it also substantially improves on runtime for problems that can be expressed. We compare runtime results for Poisson deconvolution examples with spatially-invariant 11×11 and 15×15 kernel in Table 1. The schedule used by our system involves generating an intermediate image which is the kernel convolved with the unknowns. This is more efficient than a full matrix-free implementation at mid-to-large kernel sizes. Note also that Opt [Devito et al. 2017] fails to compile when the kernel surpasses 11×11, as the matrix-free approach exceeds the compilers memory limits. At this point, our solver is over 4× faster than the one generated by Opt. We lack a GPU comparison for the spatially-variant version of this problem, but our autoscheduled solver is over 3× faster than a solver produced with the default schedule.

2 ARCHITECTURE COMPARISON

We ran our exhaustive scheduling experiment on three separate platforms. First, a desktop Linux machine with an NVIDIA TITAN X (Pascal) and an Intel Core i7-6700K Processor, which has 4 physical and 8 virtual cores. Second, a Google Compute Engine instance with a NVIDIA Tesla K80 GPU a 2 vCPUs on an Intel Broadwell processor. Third, we use an NVIDIA Jetson Nano Devkit, which is a low power embedded GPU. This final GPU has a theoretical peak bandwidth three orders of magnitude lower than the others, due to a smaller memory bus and much lower memory clock rate. In the main paper, we reported the relative slowdown of using the autoscheduler compared to using the best possible schedule on these architectures for a suite of problems. Here we repeat those results and add the absolute times for the autoscheduled solvers in Figure 6.

3 LANGUAGE DETAILS

3.1 C API

The core of the API design and energy specification draws heavily from Opt; but is streamlined to more closely mimic the graphics APIs which inspired it. The specification especially is easy to understand by analogy to an HLSL/GLSL shader.

3.2 Energy and Scheduling Language Description

The Thallo energy specification and scheduling languages are embedded in Lua. Users build up symbolic representations of the energy function through operator-overloading. Once an energy is defined, users can make method calls on the symbolic representation to provide the compiler with metadata for scheduling the energy.

There are three distinct parts of the energy specification (the inputs, the expression language where the computation is specified, and the outputs in the form of named residuals) and each has different scheduling constructs.

K80 Datacenter GPU | Titan X Desktop GPU | Jetson Nano Edge-Compute GPU

	% Off Optimum		
Example	TITAN X	K80	Nano
BundleFusion	0.00%	1.93%	1.19%
Projective Face Fit	1.67%	0.75%	4.04%
Shape and Shading	15.38%	4.46%	20.72%
Bundle Adjust	7.19%	8.30%	10.19%
Deconv. (11 × 11)	4.93%	5.74%	1.52%
Deconv. (15 × 15)	3.93%	6.95%	1.73%
SV Deconv.	0.00%	0.45%	1.81%
COT	29.81%	1.09%	2.99%

	Absolute Solve Time (Autoscheduled)		
Example	TITAN X	K80	Nano
BundleFusion	2.96ms	8.82ms	698ms
Projective Face Fit	46.3ms	253.10ms	128166ms
Shape and Shading	171.86ms	434.06ms	106610ms
Bundle Adjust	89.47ms	131.94ms	44916ms
Deconv. (11 × 11)	46.58ms	112.75ms	38072ms
Deconv. (15 × 15)	93.11ms	215.17ms	67776ms
SV Deconv.	137.92ms	319.03ms	89058ms
COT	42.11ms	115.87ms	4862ms

Fig. 6. Top Table: relative increase in solver time from using the autoscheduled solver versus the best possible one on three different platforms. Bottom Table: absolute solve times for the autoscheduled solvers across the platforms.

3.2.1 Energy Inputs. At the top of the specification, we list *Abstract Dimensions*. They are used for both, specifying the dimensions of input arrays, as well as defining Index Domains for defining what we are mapping residuals over. Using Sparse BundleFusion as an example (see Sec. 1.1 for more details of the problem statement), we have T camera poses (one for each time step t) and C correspondences.

```
T,C = Dims("T","C")
```

A key concept of the energy specification language is that of an *Index Domain*. A new Index Domain can be generated from any Dimension by calling it (`t = T()`). These Index Domains can then be used to index into both input arrays and expressions.

Next is the input block, where we declare the multidimensional arrays, scalar parameters, and sparsity structures we take as input from the CPU code using Thallo. This is roughly analogous to shader uniform blocks, used in GLSL.

```
in = Inputs {
        Cameras = Unknown(float6, {T})
        Pos_i = Array(float3, {C})
        Pos_j = Array(float3, {C})
        i = Sparse({C},{T})
        j = Sparse({C},{T})
        weightSparse = Param(float)
}
```

```
void CurveFit(int datasize, float* params, float* data) {
    Thallo_InitializationParameters init = {};
    init.doublePrecision = 0; // Single-precision only
    init.autoschedule = 1;  // Use the autoscheduler
    ThalloState* s = Thallo_NewState(init);
    // load the Thallo DSL file containing the cost description
    ThalloProblem* problem = Thallo_ProblemDefine(s,"curvefit.t");
    // describe the dimensions of the instance of the problem
    uint32_t dims[] = { 1, observation_count };
    ThalloPlan* plan = Thallo_ProblemPlan(s, problem, dims);
    // run the solver
    void* problem_data[] = { params, data };
    Thallo_ProblemSolve(s, plan, problem_data);
    double cost = Thallo_ProblemCurrentCost(s, plan);
    Thallo_PlanFree(s, plan);
    Thallo_ProblemDelete(s, problem);
    return cost;
}
```

Fig. 7. Thallo C/C++ API calls that use the curve fitting program. The highlighted line enables our autoscheduler to determine a high-performance structure for the solver without any hand optimization.

Every energy specification must have one input block declared with `Inputs{}`. The listing above shows all four different types of inputs:

- `Param` specifies a scalar, its only argument is its type.
- `Array` specifies a multidimensional array, its first argument is the scalar type, the second argument are the dimensions of the array, provided as a list of Abstract Dimensions.
- `Unknown` specifies the variables we are optimizing for. It is otherwise identical to `Array`.
- `Sparse` specifies a mapping from indices of one set of dimensions to another.

`Unknowns`, `Arrays`, and `Sparse` constructs can all be indexed into using Index Domains.

Occasionally, we do not want to optimize for all of the values in an Unknown array. For example, we may want the camera pose at time $t = 0$ to be the identity transform throughout the solve. We provide the `Exclude` construct to simplify this significantly and provide information to the compiler:

```
in.Cameras:Exclude(eq(t,0));
```

3.2.2 Expression Language. After the inputs have been declared, users can combine them to form complicated residual terms using standard scalar (or small vector/matrix) operations. Expressions are implicitly mapped over the cross-product of the dimensions of each Index Domain used to construct it.

Since Thallo's energy language is embedded in Lua, users can use standard programming features such as functions and loops when constructing their residual terms:

```
c = C()
cam_i = in.Cameras(in.i(c))
cam_j = in.Cameras(in.j(c))
TI = get_rigid_transform(cam_i)
TJ = get_rigid_transform(cam_j)
result = rigid_transform(TI,in.Pos_i(c))
            - rigid_transform(TJ,in.Pos_j(c));
```

This is the entirety of the calculation of the (unweighted) sparse term of BundleFusion [Dai et al. 2017]. c is an Index Domain for the correspondences, we use it to index into the `Sparse` constructs `in.i` and `in.j` to get the two camera indices for a given correspondence (which are indices into the dimension T). We use helper functions to extract a 4×3 matrix from each of the cameras stored as float6 vectors, and then apply another helper to get world space positions for corresponding points at `in.Pos_i(c)` and `in.Pos_i(c)`, which we subtract to get three residuals (the three components of the positional difference between the two points).

Every expression above is implicitly mapped over C, since it is the only index domain used in all of the expressions. In fact, you can index into any expression: `result(c)` is identical to `result`.

A more interesting use case of this can be seen in a simple 5×5 deconvolution example:

```
W, H, K = Dims("W", "H", "K") -- K = 5
in = Inputs {
    X      = Unknown(float, {W,H}) -- Image
    kernel = Array(float, {K,K})   -- Convolution Kernel
}
x,y,k_0,k_1 = W(), H(), K(), K()
kernel_weight = in.kernel(k_0,k_1)
pixel = in.X(x-k_0+2, y-k_1+2)
conv = Sum({k_0,k_1}, kernel_weight*pixel);
```

Here, the `kernel_weight` is implicitly mapped over $K \times K$. Then, `pixel` is implicitly mapped over $W \times H \times K \times K$. We used the `Sum()` construct to explicitly sum over both of the K dimensions, leaving us with `conv`, which is implicitly mapped over $W \times H$, and represents the result of X convolved with `kernel`.

We can then go one step further and index `conv` twice at two different locations (at a pixel and the pixel to its right), allowing us to compute a finite-difference horizontal gradient of the convolved image. This final expression is also implicitly mapped over $W \times H$.

The `Sum()` construct allows us to express residuals with a large number of intermediate terms concisely, and, since summation is associative and commutative, can expose extra dimensions of parallelism. The explicit indexing of intermediate expressions is convenient for drastically simplifying energy specifications, and serves as a heuristic clue for our autoscheduler (if a term is indexed multiple times it is likely fruitful to materialize it).

Every expression can be marked for materialization in the schedule by calling `:materialize(true)` on it. Its partial derivatives can also be materialized by calling `:materializeJ(true)`.

The `Sum()` expressions are give an extra scheduling construct, `:parallelize(true)`. When set, the compiler will parallelize applications of the Jacobian across both residuals and the terms of the sum.

Constant. A user can mark any expression as a constant (that is, not a function of the unknowns), by wrapping it in a call to `Constant()`. This causes all partial derivatives of the expression to be zero, even if it is calculated from unknowns. While our approach is mainly focused on non-linear least squares problems, this can for example be used to tackle the minimization of general ℓ_p-problems using Iteratively Reweighted Least Squares [Holland and Welsch 1977]. The key idea of IRLS is to transform a general unconstrained optimization problem to a sequence of reweighted least-squares problems by splitting the ℓ_p-norm into a quadratic and a non-quadratic part. Each of the reweighted least-squares problems can be tackled using Gauss-Newton (GN) or Levenberg-Marquard (LM). The reweighting factor is exactly the non-quadratic part and is assumed to be constant during each IRLS iteration step. This behavior can be obtained by wrapping the reweighting factor with `Constant()`.

3.2.3 Residual Definitions. The final energy function is set by constructing a `Residuals` block, which consists of a named list of expressions. Each of the named expressions is mapped over all Index Domains used within it, and referred to as a *Residual Group*.

```
e = Residuals {
        conv = expression0
        reg  = expression1
}
```

The final energy is the sum of the squares of all components of the Residual Group mapped over their Index Domains, and the majority of scheduling choices operate on Residual Groups.

Residual Groups mapped over the same Index Domains can be merged by calling `merge` on the residual block with the groups as the arguments (`merged_group = e:merge(conv,reg)`). Groups that are merged together are scheduled and computed together, enabling them to share computation for common subexpressions.

Every named Residual Group has five fields, `J`, `JtJ`, `Jp`, `JtJp`, and `JtF`. The first three of these have a function `:materialize(bool)` to control the coarse-grained materialization of algebraically relevant intermediates. `JtJp` and `JtF` are implicitly always materialized, as they are required for the solver.

Any expression or Residual Group field can have its indices reordered by the `:reorder({IndexDomains})` construct, which can be helpful for coalescing writes to avoid atomic contention.

Though not explored further in this paper, we also provide support for performing direct linear solves when materializing $\mathbf{J}^T\mathbf{J}$; the entire set of residuals can be marked for solving with `:use_direct_solve()`. This changes the numerical properties of the full solver substantially, so we do not include it in the autoscheduler.

4 CODE SIMPLICITY

One of the core design goals of Thallo is to allow users to quickly and easily write solvers for potentially complex optimization problems. Our Opt-like simple C/C++ API and design mirroring widely used shader languages helps achieve this goal. In addition, we ensure that the energy specifications themselves are expressive and concise, and that users can write high-performance schedules with relatively little code (if they forgo autoscheduling). As an admittedly rough proxy for simplicity, we compare the lines of code for all of the problems introduced in this paper and the five examples from [Devito et al. 2017] with handwritten solvers and other high-level systems (Ceres/Opt); see Table 2.

For a fair comparison, we count only energy-specific code, and specifically avoid counting the boilerplate looping code and matrix utility function implementations for Ceres and handwritten solvers. In particular, we do not count the solver structure for handwritten code, even though the structure changes with different $\mathbf{J}^T\mathbf{J}\mathbf{p}$ materialization strategies and is not shared between the different types.

All implementations are best-effort and no attempt was made to artificially shorten or lengthen the code.

In terms of simplicity, we are on par with Ceres and Opt, which are also high-level languages; however, again note the significant performance gain as evaluated in Table 1. Compared to handwritten code, our code is orders of magnitude shorter. In addition, for handwritten solvers, any change in the energy function touches code in many different places and must be manually verified, and materialization choices can have a large impact on the majority of the code, including structural code not captured by these metrics. These changes in Thallo are simple one-line scheduling constructs, and are correct by construction.

	Lines of Code				
Problem	Hand.	Ceres	Opt	Ours	Sched.
BundleFusion Sparse	216	17	15	17	2
BundleFusion Full	366	104	97	86	5
Shape-And-Shading	N/A	183	110	99	3
Face Fitting	94	54	45	33	2
Deconvolution	N/A	35	37	27	2
SV Deconvolution	N/A	37	40	29	2
ARAP Mesh Deform	210	36	18	18	0
Image Warping	280	36	21	26	2
Poisson Image Editing	67	16	13	14	2
Shape From Shading	445	193	99	99	3
Volumetric Deform	N/A	36	21	23	2

Table 2. Lines of code to implement various optimization problems in different frameworks. The rightmost column reports the number of lines of scheduling code to produce the fastest manual schedules.

5 NIST STRD

We validate solvers generated by Thallo with various schedules on all problems from the National Institute of Standards and Technology (NIST) benchmark for non-linear least squares problems, the Standard Reference Database 140[1]. We generate a Performance Profile for iteration count (as opposed to wallclock time, as in Supplement 6) with $\tau = 10^{-4}$ for six solvers generated by Thallo and one Ceres solver. The Thallo solvers are generated with three different schedules, representing different choices in coarse-grained materialization that can influence the results due to floating-point non-associativity.

Overall, all solvers, regardless of schedule, perform essentially on par with each other, with the exception of one problem. Single-precision is insufficient for our solvers to converge on this numerically challenging problem. However, the double-precision solvers generated by our method converge using roughly 100 iterations less than the equivalent Ceres solver. See Figure 8 for more details.

6 BUNDLE ADJUSTMENT EVALUATION

Bundle Adjustment is a widely studied problem. We performed an extensive comparison of Levenberg-Marquardt solvers generated by Thallo, solvers generated by Opt, and several generated by Ceres.

[1] https://www.itl.nist.gov/div898/strd/nls/nls_main.shtml

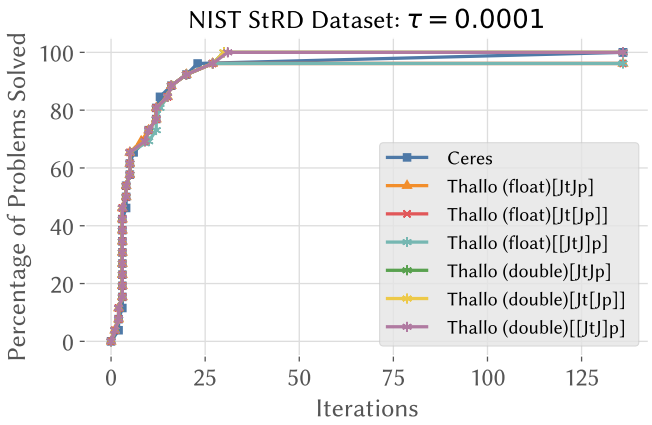

Fig. 8. Performance Profile for iteration count with $\tau = 10^{-4}$.

6.1 BAL Dataset

Bundle Adjustment in the Large[Agarwal et al. 2010b] introduced a new method for inexact Levenberg-Marquardt with a Schur preconditioner for large scale bundle adjustment problems. The authors of the paper also release a corresponding dataset, which we refer to as the BAL Dataset consisting of five different scenes with incrementally adjusted problem sizes. Altogether there are 98 different bundle adjustment problems split between them.

6.2 Performance Profiles

It is common to use *Performance Profiles* to compare the performance of a suite of optimization solvers on a large set of test problems; the resulting graphs capture robust information on solver time (or iteration count) vs. convergence (or lack thereof) [Dolan and Moré 2002]. We summarize the method here, based on the notational conventions used in Visibility Based Preconditioning for Bundle Adjustment [Kushal and Agarwal 2012].

For a given set of minimization problems $\mathcal{P}$, and a set of solvers to compare $\mathcal{S}$, run all solvers on all problems until a given convergence criteria $\mathbf{C}$ (in our examples, $\mathbf{C}$ is always a threshold wall-clock time). Then let $f(p,s)$ denote the minimum cost on problem p reached using solver s and let $f^*(p) = \min_s f(p,s)$, be the minimum cost across all solvers for problem p.

Now, for a user specified tolerance $0 < \tau < 1$, define

$$f_\tau(p) = f^*(p) + \tau(f_0(p) - f^*(p))$$

where $f_0(p)$ is the initial cost of p. Effectively, $f_\tau(p)$ is the cost at which p has been solved to within a tolerance of τ.

Let $t(p,s)$ denote the time it takes solver s to reach $f_\tau(p)$ with a value of inf if the solver fails to reach $f_\tau(p)$.

The performance profile of a solver s over the problem set P is the curve

$$\rho(s,\alpha) = 100 \times \frac{|\{p : t(p,s) < \alpha \min_s t(p,s)\}|}{|\mathcal{P}|}$$

$\rho(s, \alpha)$ measures the percentage of the problems in the suite that are solved to the tolerance τ by solver s within the time bound $\alpha \min_s t(p, s)$.

6.3 Experiments

Following[Kushal and Agarwal 2012], all the iterative solver based bundle adjustment algorithms were run inside an inexact LM loop [Wright and Holt 1985], with the forcing sequence set to a constant $\eta_k = 0.1$ and the termination rule suggested by Nash and Sofer [Nash and Sofer 1990]. We use a time budget of 60 seconds as the sole convergence criterion, C, for all problems. We compare solvers generated with Thallo (using the autoscheduler) to solvers generated with Opt, and Ceres solvers using a wide variety of preconditioners, including Schur preconditioning [Agarwal et al. 2010b] and visibility based preconditioning [Kushal and Agarwal 2012].

In Figure 9 we show performance profiles on the combined BAL dataset, consisting of 98 problems. For $\tau = 0.1$ (which is a viable threshold in an interactive pipeline), the double-precision solvers generated by both Thallo and Opt converge on a similar number of problems roughly 40 times faster than all of the Ceres solvers in our test suite. At stricter values of τ, the gap closes to approximately 10×. Thallo solvers tend to reach the convergence threshold approximately 25% faster than the equivalent Opt solver for all values of τ.

We also break down the dataset into its constituent subsets, and generate Performance Profiles for each of them in Figure 10. Here we can see that on many problems in the Ladybug dataset single-precision solvers generated with Thallo are too numerically unstable to sufficiently converge, though they are much faster for the problems they do converge for.

Note that Thallo solvers are not doing sophisticated preconditioning, which leaves numerical stability and performance on the table. In particular, the Schur Complement Trick [Agarwal et al. 2010b] is a target for future integration into Thallo: the compiler infrastructure already computes the information required to validate that an energy is of a form that can use a Schur complement in the solve (the number and kind of unknown accesses per residual).

7 COST MODEL EVALUATION

Our heuristic autoscheduler was built to quickly find a low-cost schedule among the space of all possible schedules according to the cost described in Section 5 of the paper, and as we showed, provides efficient schedules in practice across many platforms. This naturally presents a question of how predictive our simple cost model is.

In order to evaluate this, we modified our exhaustive autoscheduler experiment to output the cost model's cost as well as the time to run a single nonlinear iteration. We present the results in Figure 11, with a stochastic subset of results for applications with over 1000 schedules.

For some of the examples the cost model is highly predictive, for example the coefficient of determination (R^2) for Cotangent Mesh Smoothing is 0.99 and for Poisson Image Editing it is 0.82. The deconvolutions and Volumetric Mesh Deformation have even higher R^2 values but that appears to be driven by having 2 strong clusters of schedules, one near the chosen schedule, and a cluster of extremely expensive schedules that all are evaluated as being about as expensive as each other. On the other hand, this clustering effect also explains some of the less impressive correlations: for a clear example see the Intrinsic Image Decomposition results, where there are three distinct clusters which have strong correlations within the clusters and the appropriate relative ordering between the clusters in terms of cost, but has a nonlinear relationship between the three clusters: our cost model is accurately able to pick out the fastest cluster a roughly order the schedules correctly within clusters, but is unable to provide a direct linear relationship. This happens to an even greater degree with Face Template Fitting, which has a small cluster of extremely expensive schedules which nonetheless have a relatively low estimated cost (though significantly higher estimate cost than the cluster around the chosen schedule).

This suggests demonstrates that while our simple cost model is sufficient for acquiring good schedules on all the examples we looked at, there is plenty of opportunity for improvement by adopting a more advanced cost model, like that in the newest Halide autoscheduler Adams et al. [2019].

REFERENCES

Andrew Adams, Karima Ma, Luke Anderson, Riyadh Baghdadi, Tzu-Mao Li, Michaël Gharbi, Benoit Steiner, Steven Johnson, Kayvon Fatahalian, Frédo Durand, and Jonathan Ragan-Kelley. 2019. Learning to Optimize Halide with Tree Search and Random Programs. *ACM Trans. Graph.* 38, 4, Article 121 (July 2019), 12 pages. DOI: http://dx.doi.org/10.1145/3306346.3322967

Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. 2011. Building Rome in a Day. *Commun. ACM* 54, 10 (Oct. 2011), 105–112. DOI: http://dx.doi.org/10.1145/2001269.2001293

Sameer Agarwal, Keir Mierle, and Others. 2010a. Ceres Solver. http://ceres-solver.org. (2010).

Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. 2010b. Bundle Adjustment in the Large. In *Proceedings of the 11th European Conference on Computer Vision: Part II (ECCV'10)*. Springer-Verlag, Berlin, Heidelberg, 29–42. http://dl.acm.org/citation.cfm?id=1888028.1888032

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. 2001. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning* 3, 1 (2001), 1–122.

Antonin Chambolle and Thomas Pock. 2011. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision* 40, 1 (2011), 120–145.

Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. 2017. BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration. *ACM Trans. Graph.* 36, 3, Article 76a (May 2017). DOI: http://dx.doi.org/10.1145/3054739

Zachary Devito, Michael Mara, Michael Zollhöfer, Gilbert Bernstein, Jonathan Ragan-Kelley, Christian Theobalt, Pat Hanrahan, Matthew Fisher, and Matthias Niessner. 2017. Opt: A Domain Specific Language for Non-Linear Least Squares Optimization in Graphics and Imaging. *ACM Trans. Graph.* 36, 5, Article 171 (Oct. 2017), 27 pages. DOI: http://dx.doi.org/10.1145/3132188

Elizabeth D Dolan and Jorge J Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical programming* 91, 2 (2002), 201–213.

Mário Figueiredo and José Bioucas-Dias. 2009. Deconvolution of Poissonian images using variable splitting and augmented Lagrangian optimization. In *SWorkshop on Statistical Signal Processing*. 733–736.

Mário Figueiredo and José Bioucas-Dias. 2010. Restoration of Poissonian images using alternating direction optimization. *IEEE Trans. Image Processing* 19, 12 (2010), 3133–3145.

Felix Heide, Steven Diamond, Matthias Nießner, Jonathan Ragan-Kelley, Wolfgang Heidrich, and Gordon Wetzstein. 2016. Proximal: Efficient image optimization using proximal algorithms. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 84.

Paul W. Holland and Roy E. Welsch. 1977. Robust regression using iteratively reweighted least-squares. *Communications in Statistics – Theory and Methods* 6, 9 (Sept. 1977), 813–827. DOI: http://dx.doi.org/10.1080/03610927708827533

T. Jebara, A. Azarbayejani, and A. Pentland. 1999. 3D structure from 2D motion. *IEEE Signal Processing Magazine* 16, 3 (May 1999), 66–84. DOI: http://dx.doi.org/10.1109/79.768574

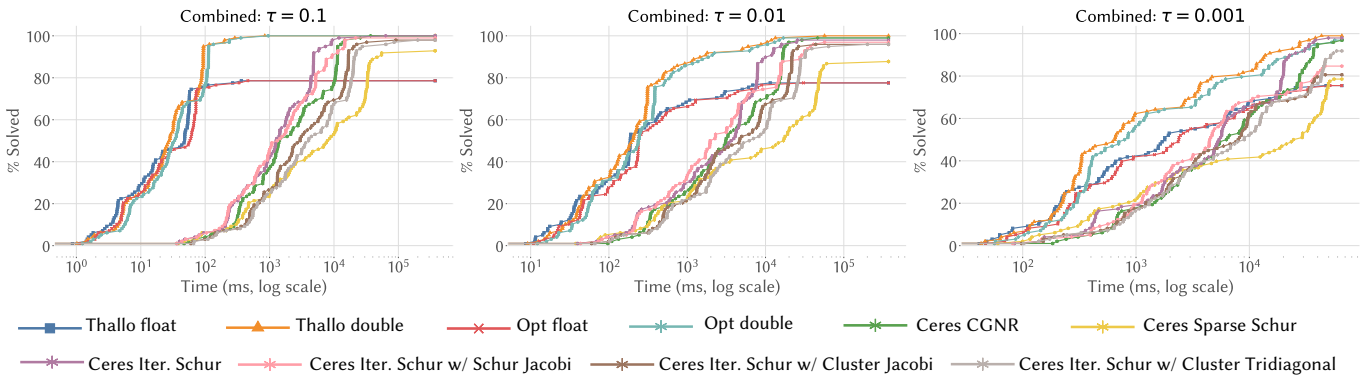

Fig. 9. Performance Profiles on the combined BAL dataset.

Dilip Krishnan, Terence Tay, and Rob Fergus. 2011. Blind deconvolution using a normalized sparsity measure. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 233–240.

Avanish Kushal and Sameer Agarwal. 2012. Visibility based preconditioning for bundle adjustment. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1442–1449.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 34, 6 (Oct. 2015), 248:1–248:16.

Ali Mosleh, Paul Green, Emmanuel Onzon, Isabelle Begin, and JM Pierre Langlois. 2015. Camera intrinsic blur kernel estimation: A reliable framework. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4961–4968.

Stephen G Nash and Ariela Sofer. 1990. Assessing a search direction within a truncated-Newton method. *Operations Research Letters* 9, 4 (1990), 219–221.

Neal Parikh and Stephen Boyd. 2013. Proximal algorithms. *Foundations and Trends in Optimization* 1, 3 (2013), 123–231.

Marc C Robini and Yuemin Zhu. 2015. Generic half-quadratic optimization for image reconstruction. *SIAM Journal on Imaging Sciences* 8, 3 (2015), 1752–1797.

Javier Romero, Dimitrios Tzionas, and Michael J. Black. 2017. Embodied Hands: Modeling and Capturing Hands and Bodies Together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)* 36, 6 (Nov. 2017), 245:1–245:17. http://doi.acm.org/10.1145/3130800.3130883

J. L. Schönberger and J. Frahm. 2016. Structure-from-Motion Revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4104–4113. DOI: http://dx.doi.org/10.1109/CVPR.2016.445

J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner. 2016. Face2Face: Real-time Face Capture and Reenactment of RGB Videos. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*.

Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. 2000. Bundle Adjustment - A Modern Synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice (ICCV '99)*. Springer-Verlag, London, UK, UK, 298–372.

SJ Wright and John Norman Holt. 1985. An inexact levenberg-marquardt method for large sparse nonlinear least squares. *The ANZIAM Journal* 26, 4 (1985), 387–403.

Chenglei Wu, Michael Zollhöfer, Matthias Nießner, Marc Stamminger, Shahram Izadi, and Christian Theobalt. 2014. Real-time Shading-based Refinement for Consumer Depth Cameras. *ACM Transactions on Graphics (TOG)* 33, 6 (2014).

Weipeng Xu, Avishek Chatterjee, Michael Zollhoefer, Helge Rhodin, Dushyant Mehta, Hans-Peter Seidel, and Christian Theobalt. 2018. MonoPerfCap: Human Performance Capture from Monocular Video. *ACM Transactions on Graphics* (2018).

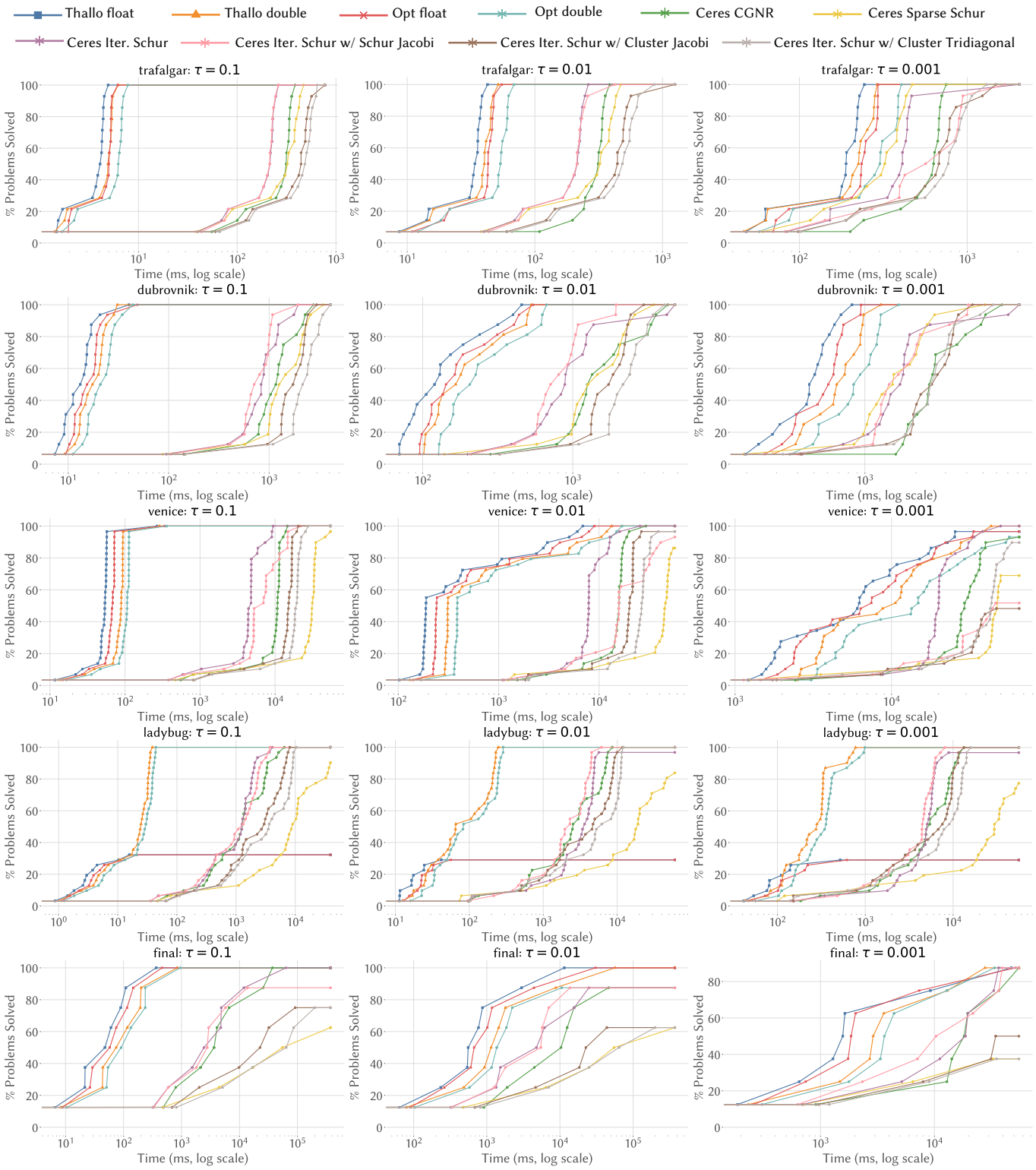

Fig. 10. Performance Profiles on the BAL sub-datasets. From top to bottom: Trafalgar, Dubrovnic, Venice, Ladybug, and Final. On the Ladybug dataset, single-precision solvers are unable to satisfactorily converge on the majority of problems, though their double-precision equivalents perform well.

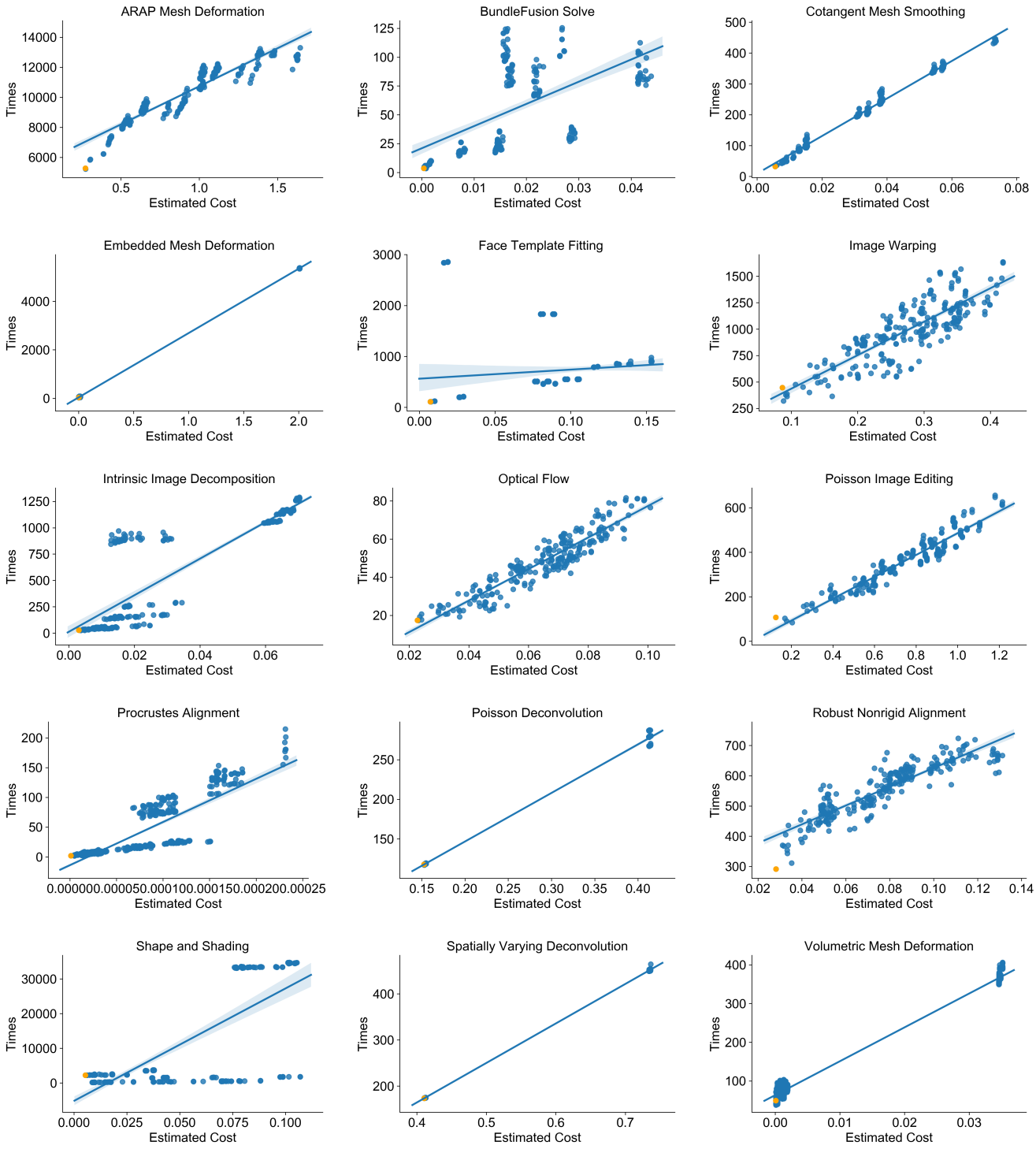

Fig. 11. For each application: on the horizontal axis is the cost model's estimated cost, on the vertical axis is the measured time to run one nonlinear iteration of the solver generated for a schedule. Each data-point corresponds to a single schedule, and for applications with over 1000 schedules we plot a stochastically chosen subset. Overlaid is the line generated by a linear regression. The minimal cost (not necessarily the fastest) schedule is highlighted in orange. For some applications this regression fits the data very well, but for several the limited fidelity of the cost model leads to a ill-fitting regression (though the minimal cost schedule is still among the fastest).