scientific reports

Check for updates

OPEN

GPU-accelerated simulated annealing based on p-bits with real-world device-variability modeling

Naoya Onizawa✉ & Takahiro Hanyu

Probabilistic computing using probabilistic bits (p-bits) presents an efficient alternative to traditional CMOS logic for complex problem-solving, including simulated annealing and machine learning. Realizing p-bits with emerging devices such as magnetic tunnel junctions introduces device variability, which was expected to negatively impact computational performance. However, this study reveals an unexpected finding: device variability can not only degrade but also enhance algorithm performance, particularly by leveraging timing variability. This paper introduces a GPU-accelerated, open-source simulated annealing framework based on p-bits that models key device variability factors-timing, intensity, and offset-to reflect real-world device behavior. Through CUDA-based simulations, our approach achieves a two-order magnitude speedup over CPU implementations on the MAX-CUT benchmark with problem sizes ranging from 800 to 20,000 nodes. By providing a scalable and accessible tool, this framework aims to advance research in probabilistic computing, enabling optimization applications in diverse fields.

In recent years, a new computational model called the probabilistic bit (p-bit) has emerged[1]. Unlike traditional bits, which are either 0 or 1, a p-bit can take any state between 0 and 1, with a probability distribution. p-bits can be implemented through software[2] or hardware such as Magnetoresistive Random Access Memory (MRAM)[3] or FPGAs[4–7]. They are particularly effective in Boltzmann machines[8], invertible logic[9], and applications like Bayesian inference[10], parallel tempering[11], Gibbs sampling[12], and simulated annealing (SA)[13–15]. SA is a stochastic optimization method[16,17] used for solving combinatorial problems like MAX-CUT[18] and enhancing machine learning algorithms[19]. These problems, often represented by Ising models, are computationally challenging (NP-hard)[20]. SA aims to minimize the 'energy' of the system, which represents the cost function of the problem. Hardware implementations of SA have also been explored for large-scale problems[21–23]. Other methods for solving Ising models include coherent Ising machines[24], simulated bifurcation[25], and coupled oscillation networks[26]. Quantum annealing (QA) promises faster solutions[27,28] but is currently limited by device performance[29,30].

SA using p-bits (pSA) is based on a probabilistic computing approach, allowing it to be implemented on conventional computers. This concept positions pSA as a promising method for tackling large-scale problems more efficiently. One of the key advantages of pSA is its ability to update multiple nodes simultaneously, unlike traditional SA, which updates nodes sequentially. This parallel update capability could accelerate the process of reaching the global minimum energy state, thereby speeding up the search for the optimal solution. Early research has shown that pSA performs well on small-scale problems[13]. However, as the problem size increases, pSA's effectiveness tends to decline. Simulations have revealed that for larger problems, such as graph isomorphism[31] and MAX-CUT, pSA struggles to maintain its efficiency[14]. In particular, the energy of the Ising model fails to decrease as expected, suggesting that pSA faces difficulties in finding optimal or near-optimal solutions for larger-scale problems.

Recently, new algorithms for pSA have been proposed to address these challenges, showing that optimal solutions can be attained even for large-scale problems[32]. These innovations have made significant strides in overcoming the limitations seen in earlier research, enhancing the overall performance of pSA in solving complex optimization tasks more efficiently. The introduction of these algorithms has opened up the possibility of large-scale implementations of pSA, which marks a major step forward in the practical application of probabilistic computing. However, achieving this goal requires careful consideration of the inherent variability in p-bit devices. Unlike traditional deterministic systems, p-bits are realized through probabilistic devices such as magnetic tunnel junction (MTJ) devices[33,34], which are subject to variations in their behavior. Recent studies have highlighted that implementations of MTJ devices, commonly used in MRAM, exhibit significant variability

Research Institute of Electrical Communication, Tohoku University, Sendai 980-8577, Japan. ✉email: naoya.onizawa.a7@tohoku.ac.jp

across different parameters[35]. This variability presents a critical challenge in the practical deployment of pSA at scale, as device-level inconsistencies can impact the overall performance and reliability of the algorithm. As such, the validation and optimization of pSA algorithms in large-scale implementations will require simulations that rigorously account for these variations. Developing accurate models that incorporate the effects of device variability is essential for understanding how pSA performs in real-world scenarios and for ensuring that large-scale p-bit systems can reliably solve complex problems.

In this article, we present a novel pSA simulator that incorporates device variability, addressing a critical challenge for future large-scale p-bit implementations. Specifically, the simulator models key types of variability, including 'timing' (device speed), 'intensity' (response to input signals), and 'offset' (shifts in input signals). By accounting for these factors, the simulator provides a more accurate representation of real-world performance in pSA systems, which is essential for assessing the viability of large-scale implementations. To handle the computational demands of simulating under various conditions of device variability, we have implemented the simulator using CUDA[36] for GPU acceleration. This allows us to efficiently simulate complex systems with numerous variability scenarios, achieving a significant two-order-of-magnitude speedup over traditional CPU-based simulations. Using the NVIDIA RTX 4090 GPU, we observed this performance boost specifically in benchmarks involving the MAX-CUT problem, demonstrating the effectiveness of GPU acceleration for such complex optimization tasks. The ability to run high-speed simulations makes it feasible to explore a wide range of conditions and configurations, which would be impractical using slower methods. Actually, we observed that device variability, especially timing variability, can lead to improved performance. In addition to its performance benefits, one of the key contributions of this work is the development of an open-source tool that can be used by the broader research community.

Methods

p-bits based simulated annealing (pSA)

p-bits and pSA. The intrinsic probabilistic nature of p-bits allows them to serve as a powerful tool for solving certain problem types that leverage randomness or uncertainty. The output state of a p-bit can be described as follows:

$$\sigma_i(t+1) = \mathrm{sgn}\Big(r_i(t) + \tanh\big(I_i(t)\big)\Big), \tag{1}$$

where $\sigma_i(t+1) \in \{-1, 1\}$ is a binary output signal, $I_i(t)$ is a real-valued input signal, and $r_i(t) \in \{-1 : 1\}$ is a random signal. The signal $r_i(t)$ is based on the p-bit model, where it is assumed to follow a uniform random distribution[1]. This assumption aligns with the theoretical formulation of p-bits, where randomness is uniformly distributed to ensure unbiased stochastic behavior. Furthermore, when implementing p-bits using hardware devices such as MTJs, the design ensures that the generated random signal closely follows a uniform distribution. This design principle guarantees consistency between theoretical models and hardware implementations.

The p-bit-based simulated annealing (pSA) method[13] is illustrated in Fig. 1. A combinatorial optimization problem is modeled using the Ising model, which represents the system's energy. This energy is governed by a Hamiltonian, expressed as follows:

$$H(\sigma) = -\sum_i h_i\sigma_i - \sum_{i<j} J_{ij}\sigma_i\sigma_j, \tag{2}$$

where $\sigma_i \in \{-1, 1\}$ represents a binary state, h denotes the biases applied to p-bits, and J signifies the connection weights between p-bits. Depending on the specific combinatorial optimization problem, different values of h and J are assigned. In pSA, each p-bit is affected by a bias h and interacts with other p-bits through the weights J. The input to each p-bit, $I_i(t)$, is calculated based on the outputs of other p-bits and is given by the following expression:

$$I_i(t) = I_0\left(h_i + \sum_j J_{ij}\cdot\sigma_j(t)\right), \tag{3}$$

where I_0 is a pseudo inverse temperature used to control the simulated annealing.

Simulated annealing seeks to find the global minimum energy of Eq. (2) by adjusting the states of σ_i. The specific method for updating σ_i varies depending on the SA algorithm employed[13–16,22]. In pSA, the pseudo inverse temperature I_0 is gradually increased to lower the energy of the Ising model. When I_0 is low, σ_i can freely flip between '-1' and '+1', enabling the algorithm to explore various potential solutions to the combinatorial optimization problem. As I_0 rises, σ_i stabilizes, steering the system toward the global minimum energy. At this stage, the values of σ_i represent the solution to the combinatorial optimization problem.

Variations of pSA. Two new pSA algorithms have been developed to tackle the challenges associated with applying pSA to large-scale combinatorial optimization problems[32]. The first of these is the time-averaged p-bit simulated annealing algorithm (TApSA), which incorporates a time-average operation into the pSA framework. This operation, as introduced in Eq. (3) of the pSA algorithm, is defined as follows:

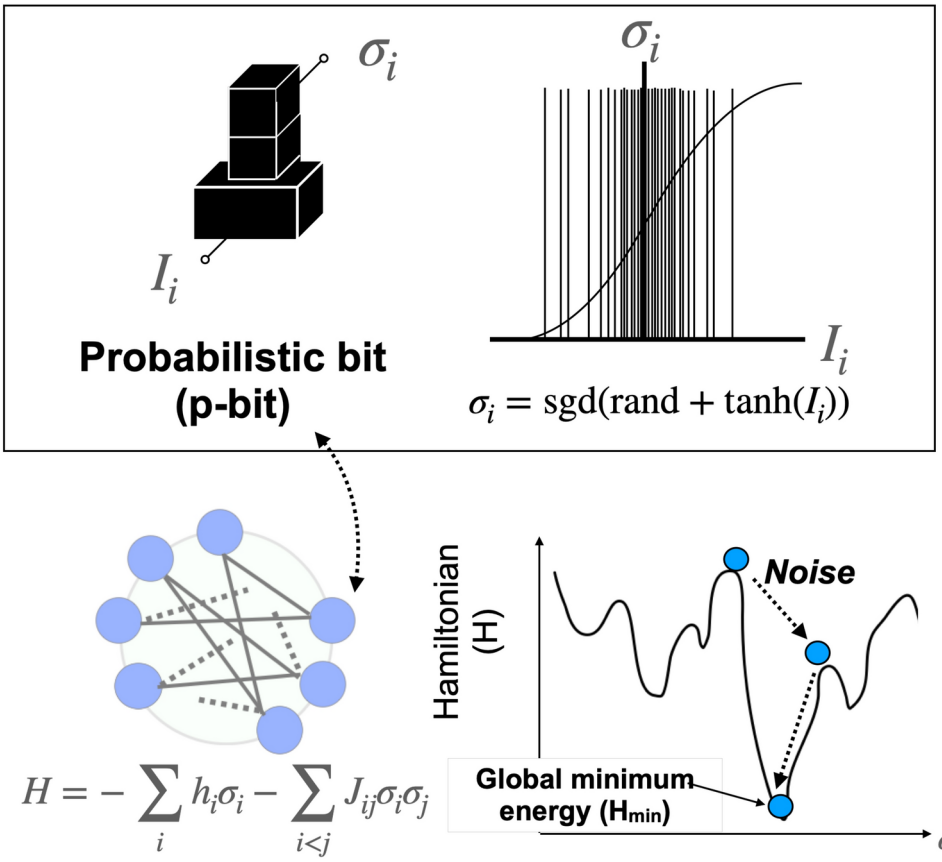

Fig. 1. Simulated annealing using p-bits (pSA) operates based on the probabilistic nature of p-bits (top), as described by Eq. (1). A combinatorial optimization problem is mapped onto an Ising model, which corresponds to an energy function (Hamiltonian). In this model, each p-bit is biased by h and interacts with other p-bits through weights J (bottom left). pSA seeks to reduce the energy of the Ising model by altering the states of p-bits σ_i. When the global minimum energy H_{min} is reached, the states σ_i represent a solution to the combinatorial optimization problem (bottom right).

$$TI_i(t) = h_i + \sum_j J_{ij} \cdot \sigma_j(t), \tag{4a}$$

$$I_i(t) = I_0 \left(\frac{1}{\alpha} \sum_{i=0}^{\alpha-1} TI_i(t-i) \right), \tag{4b}$$

where $TI_i(t)$ represents a temporary value used in the time-averaging operation, and α denotes the size of the time window over which $TI_i(t)$ is averaged. The input to the p-bit, $I_i(t)$, as defined in Eq. (1), is also applied in TApSA. The equations in Eq. (4) compute the time-averaged p-bit input signal. This averaging operation smooths the signal over a specified time window, effectively reducing random fluctuations or 'noise' in the signal.

The second algorithm is simulated annealing based on stalled p-bits (SpSA). In SpSA, the input of a p-bit, denoted as $I_i(t)$, is probabilistically stalled, retaining the value of $I_i(t-1)$ from the previous time step. The equation for SpSA, as described earlier, is given by:

$$I_i(t) = \begin{cases} I_i(t-1), & \text{with probability of getting stalled } p \\ I_0\left(h_i + \sum_j J_{ij} \cdot \sigma_j(t)\right). & \text{with probability}(1-p) \end{cases} \tag{5}$$

In this equation, the input of the p-bit at time t, $I_i(t)$, can either be stalled, meaning it remains the same as the input from the previous time step $I_i(t-1)$ with probability p, or it can take on a new value with probability $(1-p)$. This approach represents a significant departure from conventional pSA, where Eq. (3) is replaced by Eq. (5) in the SpSA algorithm.

Modeling p-bits with device variability

p-bits are realized through probabilistic devices such as MTJs[33,34]. MTJs consist of two ferromagnetic layers separated by an insulating barrier. The relative orientation of the magnetizations in these layers determines the resistance of the MTJ. When the magnetizations are parallel, the resistance is low (parallel state), and when they are antiparallel, the resistance is high (antiparallel state). This change in resistance is measured as Tunnel Magnetoresistance (TMR), which is a key parameter in the performance of p-bits.

Recently, the observed variations in MTJs primarily relate to the stochastic switching behavior between these parallel and antiparallel states, which in turn affects the performance of p-bit devices[35]. Specifically, the following variations were highlighted:

- **TMR:** TMR represents the ratio between the resistances in the parallel and antiparallel states of an MTJ. It was observed that too high a TMR value leads to undesirable plateaus in the p-bit output, where the device becomes "stuck" in one state for too long. On the other hand, too low a TMR value reduces the range of fluctuation, limiting the ability of the p-bit to explore various probabilistic states. The ideal range for TMR was found to balance between these extremes, providing a sufficient fluctuation range without causing plateaus.
- **Dwell Times:** Dwell time refers to the duration the MTJ spends in either the parallel or antiparallel resistance state before switching. This is another important source of variation. Faster MTJs, characterized by shorter dwell times, tend to produce more continuous resistance distributions, leading to smoother p-bit outputs. In contrast, slower MTJs with longer dwell times are more prone to exhibit a bimodal distribution, which can result in plateaus in the p-bit's probabilistic output.To include the device variability of p-bits, the output state of the *i*-th p-bit is updated from Eq. (1) as follows:

$$\sigma_i(t+1+\nu_i) = \mathrm{sgn}\Big(r_i(t) + \tanh\big(\lambda_i(I_i(t)+\delta_i)\big)\Big), \tag{6}$$

where three new parameters of λ_i, δ_i, and ν_i are included as shown in Fig. 2**Intensity** λ_i is a scaling parameter that adjusts the impact of the input current $I_i(t)$ on the p-bit's output. A smaller λ_i reduces the of the p-bit output to the input current, making the output fluctuations more prominent. This parameter is connected to the TMR variations of the MTJ. Large TMR variability can cause λ_i to vary, altering the switching intensity of the p-bit.

- **Offset** δ_i is a bias shift or offset parameter that controls how much the input current $I_i(t)$ deviates from its true value. δ_i can be influenced by MTJ resistance state variations (parallel and antiparallel states) or device-level offsets. For example, variability due to manufacturing processes or temperature changes can cause δ_i to vary, thereby shifting the switching probability threshold.
- **Timing** ν_i represents the delay parameter that controls when the next state update occurs. ν_i is associated with the average dwell time of the MTJ or the switching speed. Faster MTJs (with shorter dwell times) have a smaller ν_i, resulting in quicker state updates, while slower MTJs (with longer dwell times) will have a larger ν_i, indicating a delayed state update. Containing ν_i results in the need for a simulation at finer intervals than a single cycle in Eq. (6). This implies that the state σ_i must be updated at time steps smaller than the duration of one cycle. We refer to this finer subdivision of time as the time resolution of the simulation, T_{res}. By increasing the time resolution, we are able to more accurately capture the effects of device variations, thereby improving the precision of the simulation results.$\lambda_i$and δ_i should capture the TMR and energy barrier variability accurately so that the model can reflect how each p-bit's probabilistic characteristics change according to the MTJ properties. ν_i should adequately represent the switching speed of each MTJ so that the model can capture the timing variations in p-bit outputs due to MTJ dwell time variations.

The three parameters representing the device variability are modeled by normal distributions as follows:

$$\lambda_i \sim \mathcal{N}(1, \sigma_\lambda^2) \tag{7a}$$

$$\delta_i \sim \mathcal{N}(0, \sigma_\delta^2) \tag{7b}$$

$$\nu_i \sim \mathcal{N}(0, \sigma_\nu^2) \tag{7c}$$

Varying the standard deviation allows simulations under various conditions of variability.

CUDA programming and experimental conditions

The p-bit with the device variability is modeled in CUDA (Compute Unified Device Architecture)[36] for large-scale p-bit computing. CUDA is a software framework and application programming interface (API) that allows developers to use graphics processing units (GPUs) for general-purpose processing. The pseudo code of

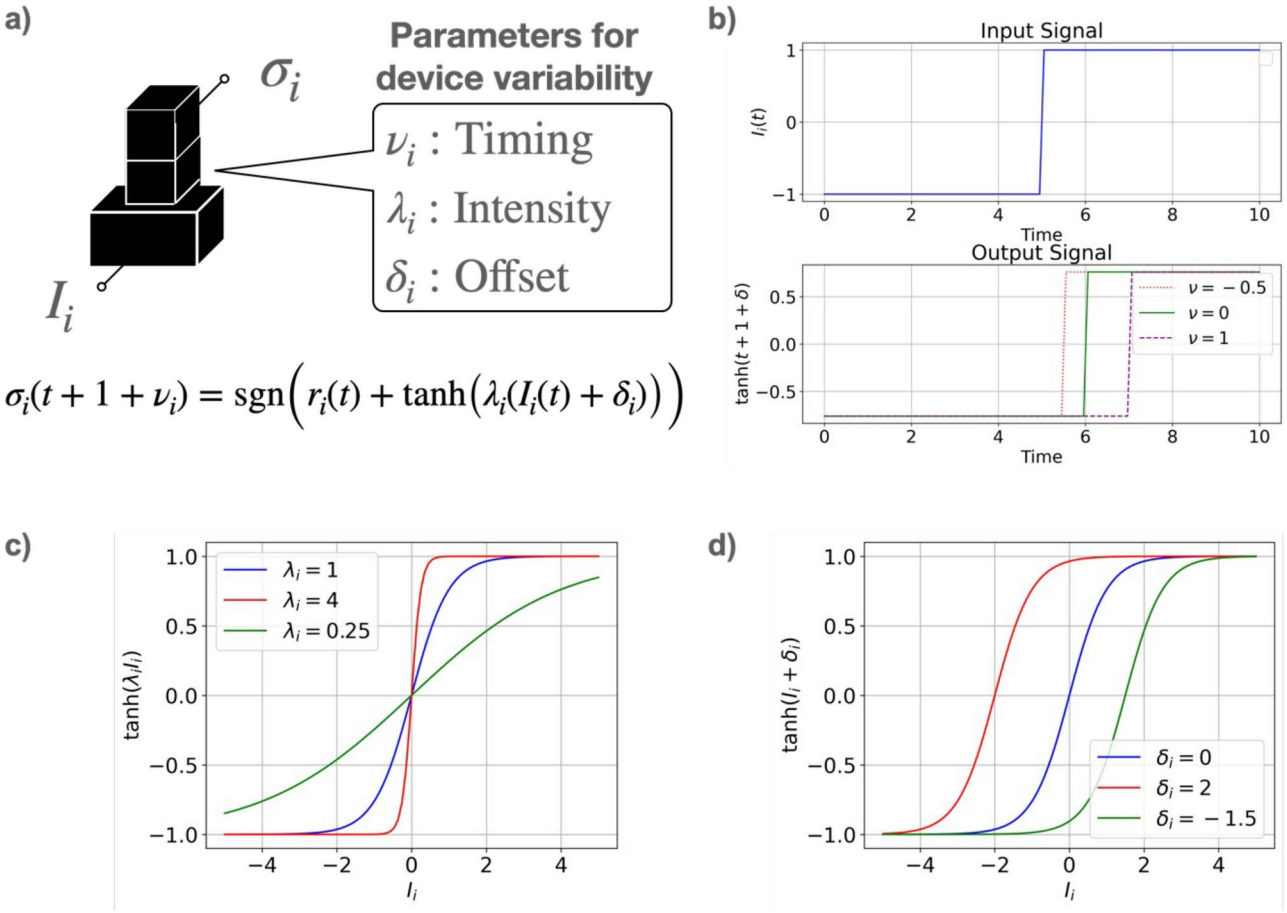

Fig. 2. Modeling of p-bits with device variability. The model introduces three new parameters to capture device variability: timing (ν_i), intensity (λ_i), and offset (δ_i). (**a**) These parameters accurately reflect the switching characteristics of each MTJ, enabling the model to account for variations in p-bit outputs due to changes in the MTJ dwell time. (**b**) The timing parameter ν_i captures shifts in the output signal in response to input signal changes. (**c**) The intensity parameter λ_i captures the variability in the steepness of the $\tanh(\lambda_i I_i)$ function, indicating intensity to input changes. (**d**) The offset parameter δ_i captures shifts in the input threshold, which affect the transition points of the output curve. These parameters combined allow the model to reflect probabilistic behavior variations in p-bits according to the properties of each MTJ.

the p-bit with the device variability is described as follows:

```
Function SimulatedAnnealingModule(vertex, mem_I0, h_vector, J_matrix, spin_vector, rnd,
lambda, delta, nu, count_device):

    For each index i in range(0, vertex):
        If (count_device mod nu[i] == 0):
            Initialize D_res with h_vector[i]

            For each index k in range(0, vertex):
                D_res = D_res + J_matrix[i][k] * spin_vector[k]

            Itanh = tanh(lambda[i] * mem_I0 * (D_res + delta[i])) + rnd[i]

            If (Itanh > 0):
                spin_vector[i] = 1
            Else:
                spin_vector[i] = -1
```

Listing 1. Pseudo code of the SimulatedAnnealingModule that updates p-bits based on device variability in CUDA. Thefunction iteratively updates the state of p-bits using a probabilistic rule involving hyperparameters such as *lambda*, *delta*, and*nu*.

This pseudo code presents a simulated annealing module that models the behavior of p-bits using variability parameters in a CUDA environment. The function `SimulatedAnnealingModule` takes in several inputs, including the number of nodes (`vertex`), the pseudo inverse temperature, I_0 (`mem_I0`), and coupling matrices (`J_matrix`). It also considers the variability parameters such as `lambda`, `delta`, and `nu`, along with a random number vector (`rnd`). The random number vector `rnd` is updated at every simulation cycle. This ensures that each probabilistic update of the spin vector is based on a new set of random values, maintaining the stochastic nature of the simulated annealing process. This CUDA code is controlled from Python using PyCUDA[37]

The simulated annealing based on the p-bit with the variability is evaluated in MAX-CUT problems. The p-bit based annealing algorithms of pSA, TApSA, and SpSA are simulated. pSA, TApSA, and SpSA compute Eqs. (3)–(5), respectively, with the device-variability p-bit model of Eq. (6). The simulations are carried out using Python 3.6 on Intel Xeon Gold 6430 and NVIDIA RTX 4090 with 128 GB of memory. In our current GPU implementation, we use 32 threads per block, where the total number of blocks is defined as $\lceil \text{number of nodes}/32 \rceil$. This configuration was chosen based on empirical testing to balance computational efficiency and memory access patterns. The relationship between thread count and GPU performance is not always linear, and increasing the number of threads does not necessarily guarantee better performance. While 32 threads have provided a good balance in our testing, we recognize that further investigation into dynamic thread allocation based on problem size could potentially improve GPU utilization. This remains an area of future exploration.

Graph	# nodes	Structure	Weights (*J*)	# edges
G1	800	Random	+1	19176
G22	2000	Random	+1	19990
G47	1000	Random	+1	9990
G48	3000	Troidal	+1, −1	6000
G55	5000	Random	+1	12498
G60	7000	Random	+1	17148
G67	10000	Troidal	+1, −1	20000
G77	14000	Troidal	+1, −1	28000
G81	20000	Troidal	+1, −1	40000

Table 1. Summary of MAX-CUT benchmarks used for evaluating simulated annealing based on p-bits with device variability. The graphs labeled as Gxx are part of the G-set benchmark set. Each graph varies in the number of nodes, structure type, edge weights (*J*), and total number of edges. These benchmarks were selected to explore the effect of variability on large-scale problem instances.

The MAX-CUT problem aims to partition a graph into two groups such that the sum of the weights of the edges connecting the two groups is maximized. This process involves cutting' the graph into two separate sections, hence the term "MAX-CUT". The MAX-CUT benchmark, namely G-set, is used for simulations (Table 1), where the G-set includes Gxx graphs that vary in node sizes and edge connections[38]. A crucial component in these pSA algorithms is the manipulation of the pseudo-inverse temperature, which plays a significant role in guiding the annealing process through the solution space. During the simulated annealing process, the pseudo-inverse temperature I_0 is gradually increased over time, starting from an initial value I_{0min} and reaching a maximum value I_{0max}, following the formula $I_0(t+1) = I_0(t)/\beta$. At smaller values of $I_0(t)$, the p-bit states are more prone to flipping, allowing for broad exploration of the solution space. As the value grows larger, the p-bit states become more stable, facilitating convergence to an optimal solution. This annealing process is central to ensuring the system transitions smoothly from exploration to exploitation. The hyperparameters for the simulated annealing processes, such as I_{0min}, I_{0max}, and β, are not arbitrarily selected. Rather, they are determined using a specific statistical method designed to optimize the performance of the simulated annealing algorithm (SSA)[39]. Similar to the previous study[32], in this simulation, I_{0min} is set to $\frac{0.1}{\text{mean}(s_i)}$ and I_{0max} is set to $\frac{10}{\text{mean}(s_i)}$, where s_i is defined as $\sqrt{(n-1)\cdot \text{Var}(J_{i,:})}$. The parameter β is calculated as $\left(\frac{I_{0min}}{I_{0max}}\right)^{\frac{1}{cycle-1}}$. Here, *n* is the number of nodes in the graph, *cycle* is the number of cycles from I_{0min} to I_{0max}, and $J_{i,:}$ is a vector containing all the edge weights connected to the *i*-th p-bit. For TApSA and SpSA, based on the results[32], the values $\alpha = 4$ and $p = 0.5$ are selected for Eqs. (4) and (5), respectively. In the studies where these algorithms were originally proposed, the parameter values were determined through extensive sweeps across a range of possible values to identify optimal settings for various problem instances. While the optimal parameter values can vary depending on the specific problem being solved, both alpha and p generally exhibit robust performance when set above certain threshold values, making them effective across a wide range of problems. $\alpha = 4$ and $p = 0.5$ were chosen as they have been shown to perform well across diverse problem sets and provide a reasonable balance between exploration and convergence. When simulating the device variability, T_{res} is set to 10 for the timing variability, ν.

To evaluate the simulation speed of the GPU implementation, a CPU version was developed using Python and executed on the same machine as the GPU implementation. In our current CPU implementation, the CPU version does not utilize parallel processing across multiple CPU cores. The performance baseline was evaluated using a single-threaded implementation in Python. Although Python provides libraries such as multiprocessing and concurrent.futures for parallel execution, enabling efficient multi-core utilization would require restructuring the CPU code, especially for tasks involving shared memory or frequent data exchange.

Results

Speed comparison of CPU and GPU

Figure 3 demonstrates the comparative performance of three simulated annealing-based algorithms-pSA, TApSA, and SpSA -executed on both CPU and GPU for a range of problem sizes without the device variability. Subplot (a) shows the scaling behavior of annealing time against the number of nodes for each algorithm. All the benchmarks in Table 1 are used with 1000 cycles. It is evident that GPU-based versions achieve significantly lower annealing times compared to their CPU counterparts, indicating the efficiency gains obtained from parallel computation. Subplot (b) illustrates the normalized mean cut values across different node sizes. The normalized mean cut value is obtained by calculating the ratio of the average cut value to the best-known cut value, with the average cut value derived from 100 trials. Both SpSA and TApSA maintain a high normalized mean cut value across all problem sizes on both CPU and GPU, highlighting their ability to find near-optimal solutions consistently. On the other hand, pSA shows a significant drop in cut value, indicating reduced effectiveness compared to the other two algorithms, as reported in the previous study[32]. Subplots (c) and (d) further analyze the mean cut value as a function of annealing time for G1 and G81, respectively, offering insights into the convergence behavior of each algorithm. The annealing time is increased when the number of cycles is increased. Subplot (c) shows that SpSA and TApSA quickly converge to higher mean cut values on GPU, reflecting their faster solution convergence with increased annealing time. Subplot (d) extends this analysis, reinforcing the observation that TApSA and SpSA efficiently leverage the GPU to achieve higher cut values, while pSA's GPU implementation struggles to compete in both efficiency and solution quality. These results underscore the effectiveness of TApSA and SpSA, particularly on GPU, in solving large-scale problems efficiently, achieving near-optimal solutions with shorter annealing times.

Simulation analysis of device variability

Figure 4 analyzes the effects of variability parameters σ_λ, σ_δ, and σ_ν for G1 on the performance of three simulated annealing algorithms: pSA, TApSA, and SpSA. The variability parameters defined by Eq. (7) can change the variability of p-bits for the input-signal intensity (λ), the input-signal offset (δ), and the update timing of p-bits (ν). The mean cut value is plotted against increasing values of each variability parameter, while the other two parameters are fixed at specific values. Each row of subplots presents a different variability parameter being varied, highlighting how the algorithms respond to changes in σ_λ, σ_δ, and σ_ν. The key observations are:

- **Robustness of TApSA and SpSA:** Across all parameter variations, TApSA and SpSA demonstrate resilience to changes in variability, maintaining consistently high mean cut values. This suggests that these algorithms are effective in achieving near-optimal solutions even under variable device conditions.
- **Sensitivity of pSA:** The pSA algorithm shows a clear sensitivity to changes in all three variability parameters, with a noticeable improvement in mean cut values as the parameters increase. It was confirmed that pSA per-

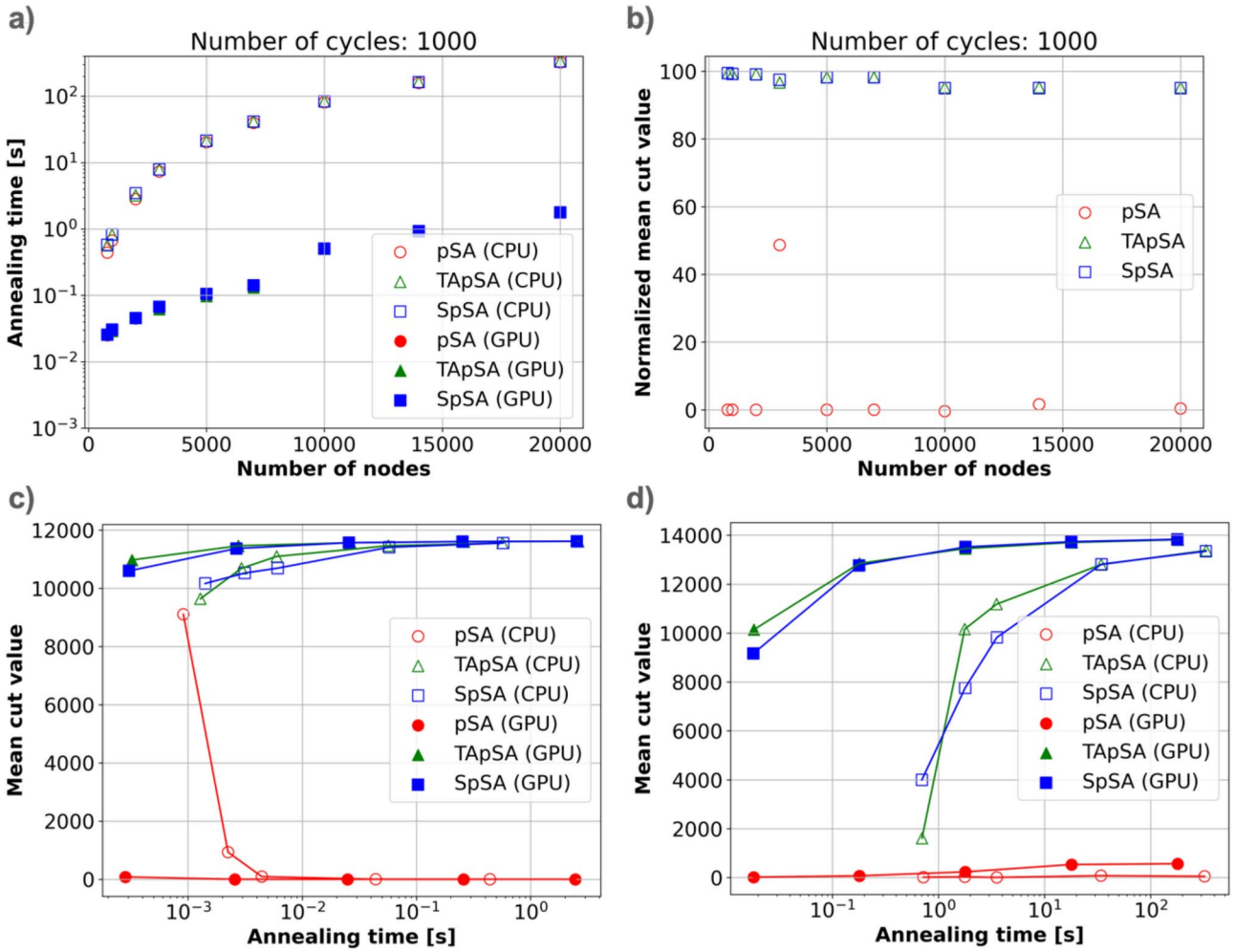

Fig. 3. Performance comparison of pSA, TApSA, and SpSA algorithms on CPU and GPU across different problem sizes without the device variability. (**a**) Annealing time as a function of the number of nodes: GPU implementations show significantly faster annealing times compared to their CPU counterparts, while pSA (GPU) exhibits longer times for larger problem sizes. (**b**) Normalized mean cut value as a function of the number of nodes: The SpSA and TApSA algorithms maintain high normalized mean cut values across all problem sizes, indicating their robustness in achieving optimal solutions. (**c**) Mean cut value versus annealing time for different algorithms in G1: As the annealing time increases, TApSA and SpSA converge to higher mean cut values, particularly in the GPU implementation. pSA struggles to find higher cut values efficiently on both CPU and GPU. (**d**) Extended analysis of mean cut value versus annealing time in G81: The GPU-based implementations of TApSA and SpSA consistently achieve higher mean cut values faster than their CPU implementations. In contrast, pSA on the GPU lags in performance relative to the other algorithms.

formance benefits from device variability, with a particularly significant impact observed from σ_ν, which represents the variability in the timing of p-bit updates. This variability implies that updates are not carried out synchronously but rather in a somewhat asynchronous manner. This de-synchronization appears to positively influence pSA performance, although it still generally falls short of TApSA and SpSA in most cases.These findings underscore the advantage of using TApSA and SpSA in applications where device variability plays a critical role, as these algorithms exhibit stability and effectiveness under varying conditions.

Figure 5 illustrates the effects of varying σ_λ, σ_δ, and σ_ν on the normalized mean cut values for three different algorithms-pSA, TApSA, and SpSA-across different node sizes. The normalized mean cut value reflects the quality of the solution relative to the optimal, with a value of 1 representing an optimal cut.

- **Resilience of SpSA and TApSA:** Across all six subplots, SpSA and TApSA demonstrate robustness to variability in σ_λ, σ_δ, and σ_ν, consistently achieving high normalized mean cut values regardless of node size. This indicates that these algorithms are well-suited for solving large-scale problems under a range of device conditions.
- **Sensitivity of pSA:** The pSA algorithm shows a much higher sensitivity to changes in the variability parameters. In all cases, pSA achieves lower normalized mean cut values, especially as the number of nodes increases.

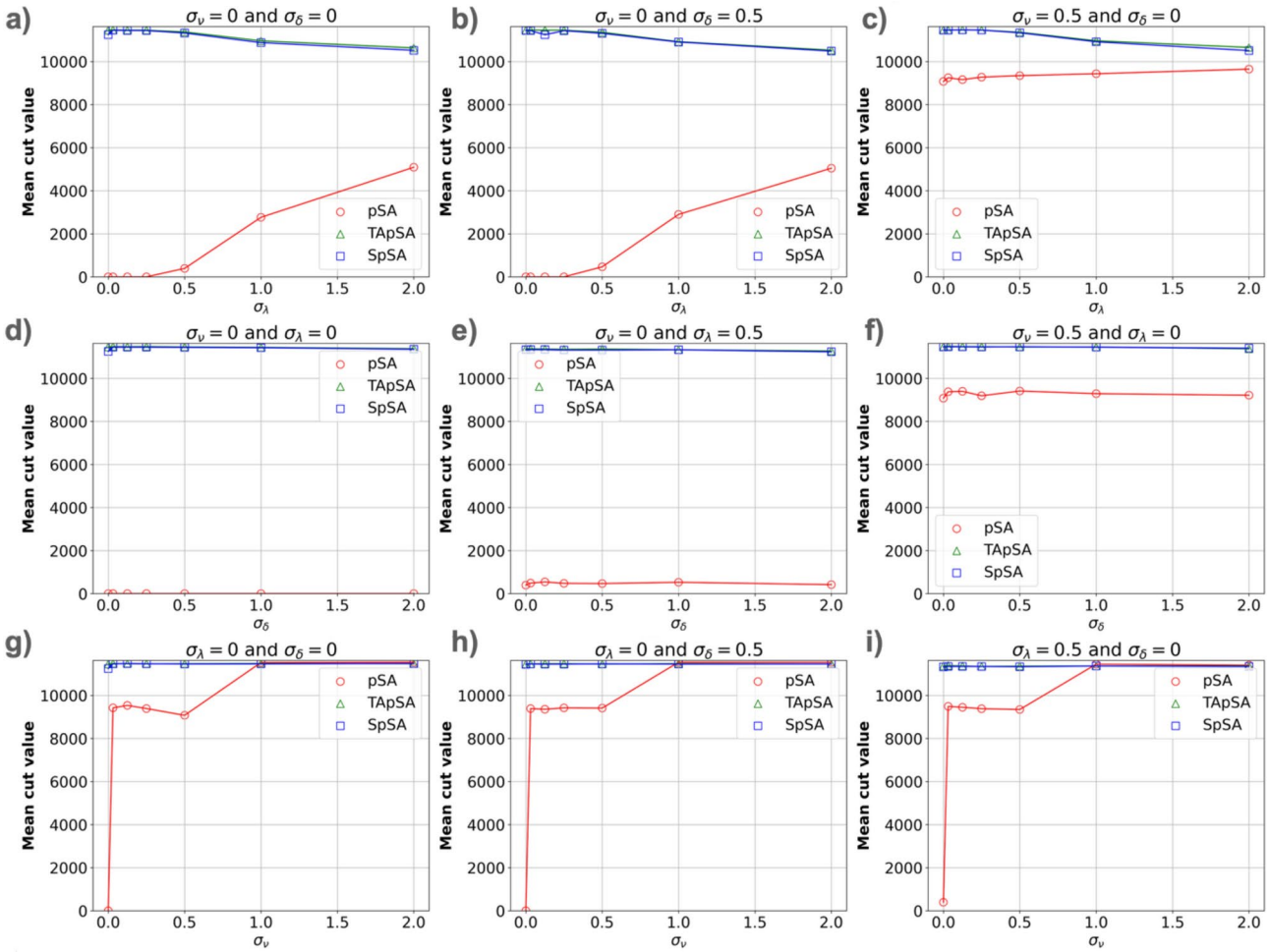

Fig. 4. Impact of variability parameters on mean cut value for different simulated annealing algorithms for G1. Each subplot shows the sensitivity analysis of three algorithms (pSA, TApSA, and SpSA) under variations in key variability parameters: σ_λ, σ_δ, and σ_ν. (**a**–**c**) Effects of σ_λ under different combinations of σ_δ and σ_ν. When both $\sigma_\delta = 0$ and $\sigma_\nu = 0$, the mean cut values for pSA improve steadily with increasing σ_λ, while TApSA and SpSA maintain consistently high values (**a**). As σ_δ increases to 0.5 (**b**), the effect on pSA's mean cut values diminishes slightly. The trends remain stable when σ_ν increases to 0.5 (**c**), showing resilience in TApSA and SpSA. Notably, in case (**c**), pSA also achieves a high mean cut value, indicating its robustness under these conditions. (**d**–**f**) Effects of σ_δ under different combinations of σ_λ and σ_ν. With $\sigma_\lambda = 0$ and $\sigma_\nu = 0$ (**d**), the pSA shows a gradual improvement in mean cut values with increasing σ_δ, while TApSA and SpSA maintain stable high performance. When σ_λ increases to 0.5 (**e**), the patterns remain similar. As σ_ν increases to 0.5 (**f**), the stability in SpSA and TApSA continues. (**g**–**i**) Effects of σ_ν under different combinations of σ_λ and σ_δ. For $\sigma_\lambda = 0$ and $\sigma_\delta = 0$ (**g**), the impact on pSA is evident, with an initial increase in mean cut values. As σ_λ increases to 0.5 (**h**), the stability of TApSA and SpSA remains unaffected, while pSA shows a slight decrease in mean cut values. When σ_δ increases to 0.5 (**i**), all algorithms demonstrate high stability, with SpSA consistently performing the best.

This suggests that pSA is less capable of finding optimal or near-optimal solutions when subject to device variability, particularly when the problem size becomes large.These results highlight the superior performance of SpSA and TApSA under varying device conditions, making them more reliable choices for large-scale optimization tasks where device variability may impact the solution quality.

Discussion

To investigate in detail how device variability influences normalized mean cut values, we examined the energy changes during annealing. Figure 6 presents the impact of varying device variability parameters (σ_λ, σ_δ, and σ_ν) on energy evolution during simulated annealing cycles for G1. The subplots (a), (c), and (e) are obtained by pSA, and the others are obtained by TApSA. Since the performance of TApSA and SpSA is nearly identical, only TApSA was observed in this analysis. Each subplot explores how different variability conditions affect the annealing process, which aims to reach a stable, low-energy configuration.

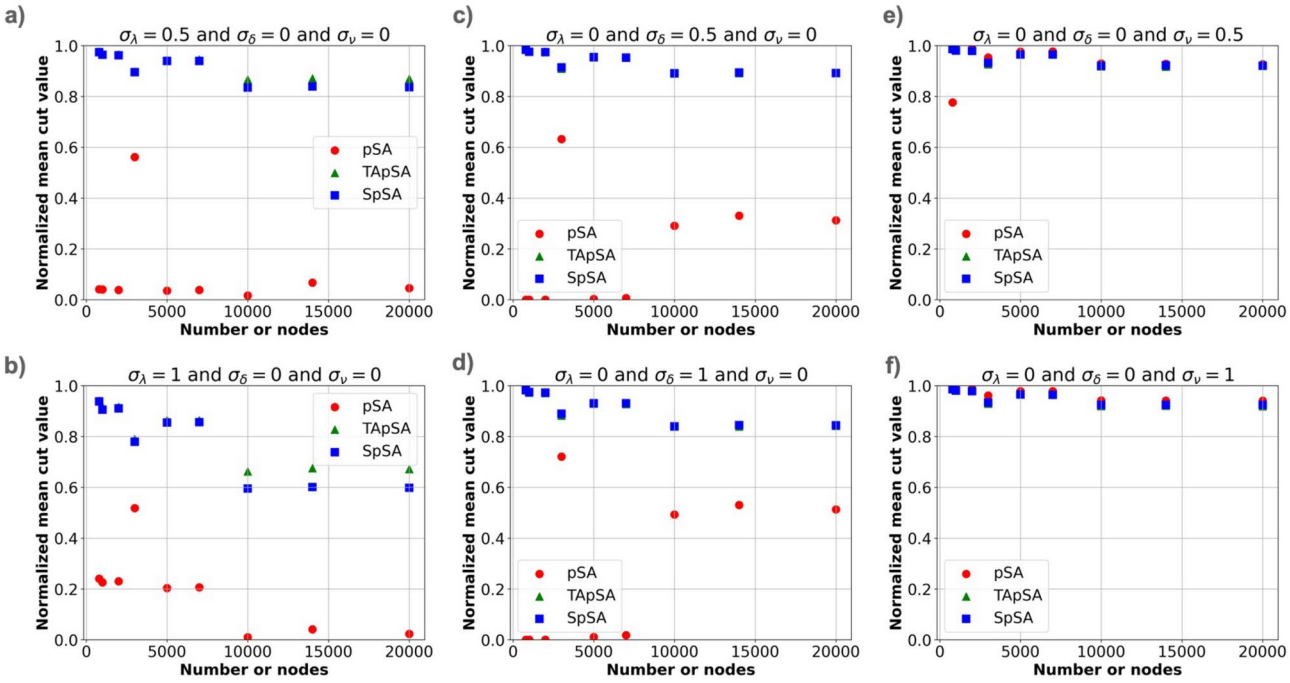

Fig. 5. Impact of variability in σ_λ, σ_δ, and σ_ν on the normalized mean cut value for different node sizes. Each subplot shows how the normalized mean cut value changes with increasing number of nodes under various levels of variability for pSA, TApSA, and SpSA. (**a**) $\sigma_\lambda = 0.5$, $\sigma_\delta = 0$, and $\sigma_\nu = 0$: SpSA and TApSA maintain high normalized mean cut values across all node sizes, while pSA shows lower cut values with a significant drop at higher node counts. (**b**) $\sigma_\lambda = 1$, $\sigma_\delta = 0$, and $\sigma_\nu = 0$: As σ_λ increases to 1, SpSA continues to perform well, but TApSA shows a small decrease in performance, and pSA remains low. (**c**) $\sigma_\lambda = 0$, $\sigma_\delta = 0.5$, and $\sigma_\nu = 0$: Variability in σ_δ affects pSA more strongly, reducing its cut values, while TApSA and SpSA remain largely unaffected. (**d**) $\sigma_\lambda = 0$, $\sigma_\delta = 1$, and $\sigma_\nu = 0$: Further increase in σ_δ results in little to no effect on SpSA and TApSA, while pSA still performs poorly. (**e**) $\sigma_\lambda = 0$, $\sigma_\delta = 0$, and $\sigma_\nu = 0.5$: Introducing variability in σ_ν shows resilience in both SpSA and TApSA across all node sizes, but pSA remains significantly affected. (**f**) $\sigma_\lambda = 0$, $\sigma_\delta = 0$, and $\sigma_\nu = 1$: Even under high variability in σ_ν, SpSA and TApSA maintain high normalized cut values, whereas pSA remains sensitive and continues to increase the normalized mean cut values.

- **Effect of σ_λ:** In plots (a) and (b), as σ_λ increases, the system exhibits higher final energy states, indicating difficulty in finding low-energy solutions. In the minimization-focused plot (b), increased σ_λ causes slower convergence and less effective energy minimization, emphasizing that higher σ_λ disrupts stability.
- **Effect of σ_δ:** Plots (c) and (d) show similar trends with σ_δ. Higher variability in σ_δ has minimal impact on the final energy state and does not significantly affect annealing efficiency, as shown in (d) with varying σ_δ values.
- **Effect of σ_ν:** In plots (e) and (f), increasing σ_ν in (e) makes it easier to reach low-energy states, indicating that greater variability in σ_ν can enhance convergence. In contrast, plot (f) shows that convergence to low-energy states is minimally affected by σ_ν variability, maintaining stable performance.The analysis highlights how variability in parameters σ_λ, σ_δ, and σ_ν impacts the annealing process. An increase in σ_λ generally leads to higher final energy states and reduced convergence stability, making it challenging to reach low-energy solutions in TApSA. In contrast, σ_δ shows minimal effect on both energy states and annealing efficiency, indicating robustness against its variability in both pSA and TApSA. Interestingly, greater variability in σ_ν can facilitate convergence to low-energy states under certain conditions in pSA, suggesting that σ_ν variability may enhance the annealing process in some cases.

In our previous work, we observed that in the standard pSA implementation, p-bit states tended to oscillate, preventing the system from effectively reducing energy and reaching an optimal solution[32]. This issue likely arose because all p-bits were updated simultaneously, creating a synchronized behavior across the system that hindered proper state transitions. With the introduction of timing variability, the update timings of individual p-bits became slightly misaligned. This desynchronization appears to have mitigated the oscillation issue, allowing the system to explore the solution space more effectively and converge towards optimal or near-optimal solutions. It can be hypothesized that the slight asynchronicity introduced by timing variability plays a beneficial role in preventing undesired synchronization effects, thereby improving the overall efficiency of the SA process.

In this study, we conducted a comprehensive performance analysis of three simulated annealing algorithms-pSA, TApSA, and SpSA-under varying device variability conditions. We evaluated the impact of variability parameters σ_λ, σ_δ, and σ_ν on annealing time, mean cut value, and energy evolution across different problem sizes. Our findings show that SpSA and TApSA consistently outperform pSA, particularly for large-scale optimization problems, with resilience to changes in variability parameters. SpSA, in particular, demonstrates

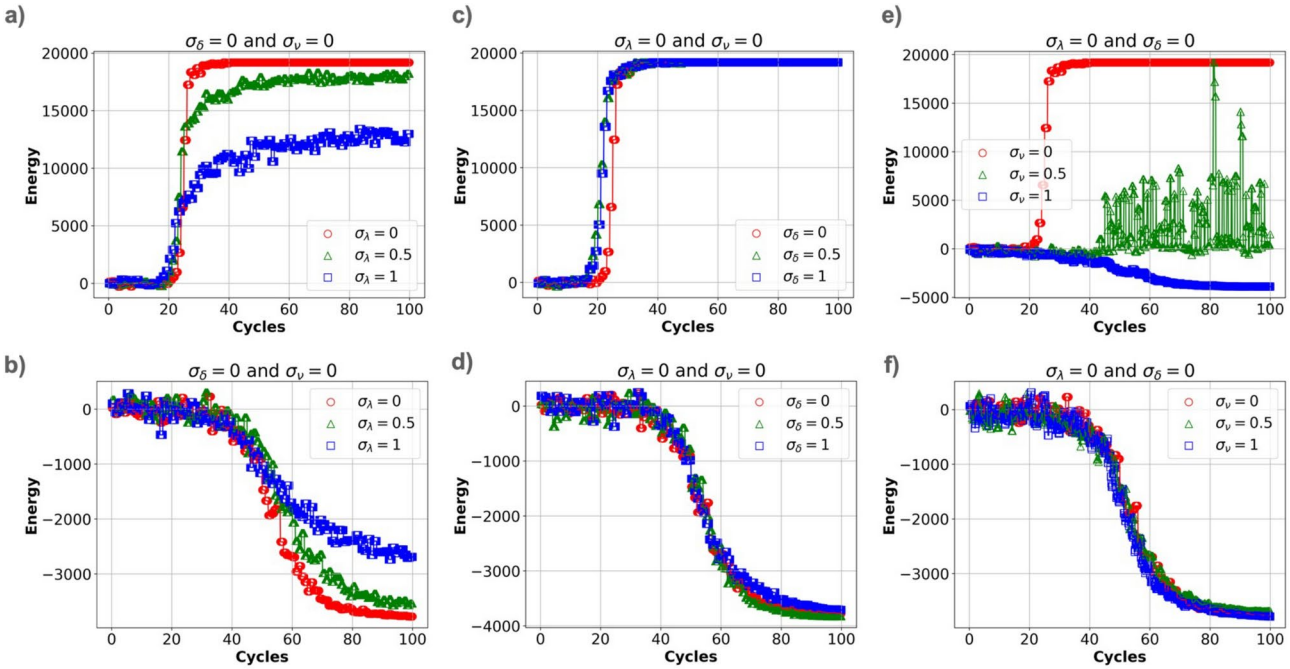

Fig. 6. Energy evolution across simulated annealing cycles under varying levels of σ_λ, σ_δ, and σ_ν for G1. The subplots (**a**, **c**, and **e**) are obtained by pSA and the others are obtained by TApSA. Each subplot shows the energy trajectory over 100 cycles with different variability conditions applied to specific parameters. (**a**): $\sigma_\delta = 0$, $\sigma_\nu = 0$, and varying σ_λ values (0, 0.5, 1). Higher σ_λ values lead to elevated final energy levels, indicating improved performance in finding low-energy solutions as variability in σ_λ increases. (**b**): $\sigma_\delta = 0$, $\sigma_\nu = 0$, and varying σ_λ, focusing on systems optimized to minimize energy. Increased σ_λ results in slower convergence toward lower energy states, with less effective minimization as σ_λ rises. (c): $\sigma_\lambda = 0$, $\sigma_\nu = 0$, and varying σ_δ (0, 0.5, 1). In (c), the results show minimal impact from variability in σ_δ, with little effect on the final energy values. (**d**): $\sigma_\lambda = 0$, $\sigma_\nu = 0$, focusing on energy minimization with varying σ_δ. The results show that as σ_δ increases, the system's ability to reach low-energy states slightly diminishes. (**e**): $\sigma_\lambda = 0$, $\sigma_\delta = 0$, with varying σ_ν (0, 0.5, 1). At $\sigma_\nu = 0.5$, energy behavior is unstable, while at $\sigma_\nu = 1$, energy decreases steadily, reaching a near-optimal state. (**f**): $\sigma_\lambda = 0$, $\sigma_\delta = 0$, focusing on energy minimization with varying σ_ν. The results show minimal impact from changes in σ_ν, with no significant difference in convergence performance.

the highest efficiency and robustness, making it well-suited for real-world applications. In contrast, pSA is highly sensitive to variability, especially for larger problems, limiting its competitiveness. Unexpectedly, we observed that device variability can both degrade and enhance algorithm performance. While it was anticipated that variability might lead to instability, particularly for pSA, we found that specific types of variability, such as timing variability, can actually improve algorithm efficiency in some cases. This dual effect highlights the complex role of device variability in p-bit-based simulated annealing and provides new insights into optimizing performance through controlled variability. Additionally, we developed a novel pSA simulator that incorporates device variability to address challenges in large-scale p-bit implementations. Using CUDA for GPU acceleration, we achieved a significant speedup, enabling simulations under diverse variability conditions. In energy evolution, SpSA and TApSA quickly converge to low-energy states, even under high variability, whereas pSA struggles with stability. This open-source simulator provides the research community with a valuable tool for exploring these algorithms and optimizing their performance for practical applications.

We would like to clarify that the main focus of this paper is not on benchmarking computational speed between the CPU and GPU implementations or comparing performance with alternative optimization algorithms. Instead, our primary objective is to demonstrate the impact of device variability (e.g., timing, parameter fluctuations) on p-bit implementations and to provide an open-source GPU-accelerated pSA simulator. To validate our approach, we selected the MAX-CUT problem, a widely recognized benchmark for combinatorial optimization problems. While CPU-GPU speed comparisons are included as a supplementary analysis to illustrate the acceleration capabilities, they are not the central focus of our study. We also recognize the importance of evaluating our simulator with other benchmark problems in future work to further validate its versatility and effectiveness across a broader range of combinatorial optimization challenges. Additionally, while a comparative analysis with other solvers could provide valuable insights, it falls outside the scope of this paper and may be considered as part of future research directions.

In terms of energy efficiency on optimization hardware, FPGA implementations of p-bit models would likely offer better energy efficiency compared to our GPU-based pSA simulator[4]. However, our primary focus in this work is on developing a high-speed pSA simulator using GPUs to evaluate the impact of device variability. This

simulator is intended as a precursor to hardware implementations that leverage future p-bit devices, such as MTJ-based devices. If p-bit devices can be realized as actual hardware, it is expected that their energy efficiency would surpass that of traditional FPGA or ASIC implementations[3]. This is due to the inherently low-power nature of p-bit devices and their suitability for probabilistic computation. Consequently, while the current GPU implementation is not optimized for energy efficiency, it serves as a critical step toward evaluating and designing energy-efficient pSA hardware using real p-bit devices in the future.

Data availability

All data generated or analyzed during this study are included in this published article. The Python codes are available at https://github.com/nonizawa/GPU-pSAv[40].

Received: 6 November 2024; Accepted: 13 February 2025
Published online: 19 February 2025

References

1. Camsari, K., Faria, R., Sutton, B. & Datta, S. Stochastic p-bits for invertible logic. *Phys. Rev. X* **7**, 031014 (2017).
2. Pervaiz, A. Z., Ghantasala, L. A., Camsari, K. Y. & Datta, S. Hardware emulation of stochastic p-bits for invertible logic. *Sci. Rep.* **7**, 10994. https://doi.org/10.1038/s41598-017-11011-8 (2017).
3. Borders, W. A. et al. Integer factorization using stochastic magnetic tunnel junctions. *Nature* **573**, 390–393. https://doi.org/10.1038/s41586-019-1557-9 (2019).
4. Pervaiz, A. Z., Sutton, B. M., Ghantasala, L. A. & Camsari, K. Y. Weighted p -bits for FPGA implementation of probabilistic circuits. *IEEE Trans. Neural Netw. Learn. Syst.* **30**, 1920–1926 (2019).
5. Smithson, S. C., Onizawa, N., Meyer, B. H., Gross, W. J. & Hanyu, T. Efficient CMOS invertible logic using stochastic computing. *IEEE Trans. Circuits Syst. I Regul. Pap.* **66**, 2263–2274 (2019).
6. Sutton, B. et al. Autonomous probabilistic coprocessing with petaflips per second. *IEEE Access* **8**, 157238–157252 (2020).
7. Aadit, N. A., Grimaldi, A., Finocchio, G. & Camsari, K. Y. Physics-inspired ising computing with ring oscillator activated p-bits, in *2022 IEEE 22nd International Conference on Nanotechnology (NANO)* 393–396 (2022).
8. Hinton, G. E., Sejnowski, T. J. & Ackley, D. H. *Boltzmann Machines: Constraint Satisfaction Networks that Learn. Tech. Rep. CMU-CS-84-119, Department of Computer Science, Carnegie-Mellon University* (1984).
9. Onizawa, N., Smithson, S. C., Meyer, B. H., Gross, W. J. & Hanyu, T. In-hardware training chip based on CMOS invertible logic for machine learning. *IEEE Trans. Circuits Syst. I Regul. Pap.* **67**, 1541–1550 (2020).
10. Kaiser, J. et al. Hardware-aware in situ learning based on stochastic magnetic tunnel junctions. *Phys. Rev. Appl.* **17**, 014016 (2022).
11. Grimaldi, A. et al. Spintronics-compatible approach to solving maximum-satisfiability problems with probabilistic computing, invertible logic, and parallel tempering. *Phys. Rev. Appl.* **17**, 024052. https://doi.org/10.1103/PhysRevApplied.17.024052 (2022).
12. Aadit, N. A. et al. Massively parallel probabilistic computing with sparse Ising machines. *Nat. Electron.* **5**, 460–468. https://doi.org/10.1038/s41928-022-00774-2 (2022).
13. Camsari, K. Y., Sutton, B. M. & Datta, S. p-bits for probabilistic spin logic. *Appl. Phys. Rev.* **6**, 011305 (2019).
14. Onizawa, N., Katsuki, K., Shin, D., Gross, W. J. & Hanyu, T. Fast-converging simulated annealing for Ising models based on integral stochastic computing. *IEEE Trans. Neural Netw. Learn. Syst.* **34**, 1–7 (2022).
15. Onizawa, N., Sasaki, R., Shin, D., Gross, W. J. & Hanyu, T. Stochastic simulated quantum annealing for fast solution of combinatorial optimization problems. *IEEE Access* **12**, 102050–102060 (2024).
16. Kirkpatrick, S., Gelatt, C. D. Jr. & Vecchi, M. P. Optimization by simulated annealing. *Science* **220**, 671–680 (1983).
17. Johnson, D. S., Aragon, C. R., McGeoch, L. A. & Schevon, C. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Oper. Res.* **39**, 378–406 (1981).
18. Mykleburst, T. Solving maximum cut problems by simulated annealing. *CoRR*[SPACE]arXiv:1505.03068 (2015). arXiv:1110.2574
19. Park, H.-K., Lee, J.-H., Lee, J. & Kim, S.-K. Optimizing machine learning models for granular NDFEB magnets by very fast simulated annealing. *Sci. Rep.* **11**, 3792. https://doi.org/10.1038/s41598-021-83315-9 (2021).
20. Reiter, E. E. & Johnson, C. M. *Limits of Computation: An Introduction to the Undecidable and the Intractable* (Chapman and Hall/CRC, 2012).
21. Aramon, M. et al. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Front. Phys.*[SPACE]https://doi.org/10.3389/fphy.2019.00048 (2019).
22. Gyoten, H., Hiromoto, M. & Sato, T. Enhancing the solution quality of hardware ising-model solver via parallel tempering, in *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '18 (Association for Computing Machinery, New York, NY, USA, 2018). https://doi.org/10.1145/3240765.3240806
23. Shin, D., Onizawa, N., Gross, W. J. & Hanyu, T. Memory-efficient FPGA implementation of stochastic simulated annealing. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **13**, 108–118 (2023).
24. Wang, Z., Marandi, A., Wen, K., Byer, R. L. & Yamamoto, Y. Coherent Ising machine based on degenerate optical parametric oscillators. *Phys. Rev. A* **88**, 063853. https://doi.org/10.1103/PhysRevA.88.063853 (2013).
25. Goto, H., Tatsumura, K. & Dixon, A. R. Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems. *Sci. Adv.* **5**, eaav2372. https://doi.org/10.1126/sciadv.aav2372 (2019).
26. Dutta, S. et al. An Ising Hamiltonian solver based on coupled stochastic phase-transition nano-oscillators. *Nat. Electron.* **4**, 502–512. https://doi.org/10.1038/s41928-021-00616-7 (2021).
27. Kadowaki, T. & Nishimori, H. Quantum annealing in the transverse Ising model. *Phys. Rev. E* **58**, 5355–5363 (1998).
28. Boixo, S. et al. Evidence for quantum annealing with more than one hundred qubits. *Nat. Phys.* **10**, 218–224. https://doi.org/10.1038/nphys2900 (2014).
29. Zick, K. M., Shehab, O. & French, M. Experimental quantum annealing: Case study involving the graph isomorphism problem. *Sci. Rep.* **5**, 11168. https://doi.org/10.1038/srep11168 (2015).
30. Yarkoni, S., Raponi, E., Bäck, T. & Schmitt, S. Quantum annealing for industry applications: introduction and review. *Rep. Progress Phys.* **85**, 104001. https://doi.org/10.1088/2F1361-6633/2Fac8c54 (2022).
31. Grohe, M. & Neuen, D. Recent advances on the graph isomorphism problem. *CoRR*[SPACE]arXiv:2011.01366 (2020).
32. Onizawa, N. & Hanyu, T. Enhanced convergence in p-bit based simulated annealing with partial deactivation for large-scale combinatorial optimization problems. *Sci. Rep.* **14**, 1339. https://doi.org/10.1038/s41598-024-51639-x (2024).
33. Hayakawa, K. et al. Nanosecond random telegraph noise in in-plane magnetic tunnel junctions. *Phys. Rev. Lett.* **126**, 117202. https://doi.org/10.1103/PhysRevLett.126.117202 (2021).
34. Safranski, C. et al. Demonstration of nanosecond operation in stochastic magnetic tunnel junctions. *Nano Lett.* **21**, 2040–2045 (2021).
35. Daniel, J. et al. Experimental demonstration of an on-chip p-bit core based on stochastic magnetic tunnel junctions and 2d mos2 transistors. *Nat. Commun.* **15**, 4098. https://doi.org/10.1038/s41467-024-48152-0 (2024).

36. Ghorpade, J., Parande, J., Kulkarni, M. & Bawaskar, A. Gpgpu processing in cuda architecture. *Adv. Comput. Int. J. (ACIJ)* (2012).
37. Klöckner, A. et al. Pycuda: GPU run-time code generation for high-performance computing. *CoRR*[SPACE]arXiv:0911.3456 (2009).
38. Ye, Y. *Computational Optimization Laboratory* (1999). http://web.stanford.edu/~yyye/Col.htm
39. Onizawa, N., Kuroki, K., Shin, D. & Hanyu, T. Local energy distribution based hyperparameter determination for stochastic simulated annealing. *IEEE Open J. Signal Process.* **4**, 452–461 (2023).
40. Onizawa, N. *Gpu-psav: A GPU-based Simulated Annealing Algorithm for p-bit Devices with Device Variability* (accessed 18 October 2024). https://github.com/nonizawa/GPU-pSAv

Acknowledgements
This work was supported in part by JST CREST Grant Number JPMJCR19K3, JSPS KAKENHI Grant Number JP21H03404, and KIOXIA Corporation.

Author contributions
N. O. conducted and analyzed the experiments. T. H. discussed the experiment. All authors reviewed the manuscript.

Additional information

Competing financial interests:
The authors declare no competing financial interests.

Additional information
Correspondence and requests for materials should be addressed to N.O.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.